A FUZZY LOGIC APPROACH TO LOAD BALANCING IN AUGMENTED REALITY DISTRIBUTED ENVIRONMENTS

APPROVED BY SUPERVISING COMMITTEE:

_____
David Akopian, Ph.D., Chair

_____
Mo Jamshidi, Ph.D., Co-Chair

_____
Wei-Ming Lin, Ph.D.

_____
Sevki Erdogan, Ph.D.

Accepted: _____
Dean, Graduate School

DEDICATION

This dissertation is dedicated to my family.

A FUZZY LOGIC APPROACH TO LOAD BALANCING IN AUGMENTED REALITY DISTRIBUTED ENVIRONMENTS

by

ALEKSANDR V. PANCHUL, M.S.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Electrical and Computer Engineering
December 2010

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ACKNOWLEDGEMENTS

Quantitative analysis in this thesis would not exist without Computer Engineering Framework (CEF) which was influenced by many people. To avoid forgetting anyone, I will leave them largely unnamed. I would like to express my deepest appreciation to all supporters of CEF projects, people who donated their computers and hardware, and whose motivation helped me to stay focused on my work and to see the bigger picture.

I would like to thank Dr. David Akopian and Dr. Mo Jamshidi, my supervising professors, whose mentoring helped me to survive in academic world. I am grateful to UTSA Department of Electrical and Computer Engineering for the given opportunities. I also would like to express my genuine gratitude to the committee for their constructive criticism and useful feedback.

I would like to acknowledge Polley-Kane and Associates, Inc. for the financial support of the supercomputer system implementation, especially in connection with the environmental research.

It is with great pleasure I would like to acknowledge Dr.Vladimir Petrukhin, professor at Moscow Institute of Physics and Technology, Russia, Kiev Institute of Cybernetics, Ukraine, and International entrepreneur. He was my mentor during undergraduate years; he convinced me to pursue formal education and helped to stay on track.

Last but not least, I would like to thank my wife, Sarah Kane Panchul for occasional proof-reading and for tolerating fifty computer servers and equipment in our small apartment.

December 2010

A FUZZY LOGIC APPROACH TO LOAD BALANCING IN AUGMENTED REALITY DISTRIBUTED ENVIRONMENTS

Aleksandr Panchul, Ph.D.
The University of Texas at San Antonio, 2010

Supervising Professors: David Akopian, Ph.D. and Mo Jamshidi, Ph.D.

By the end of the 20th century the Gordon Moor's Law could not be fulfilled by the industry in its straight form. His widely known estimation predicted in 1965 that the number of transistors in the minimum-cost CPU would double every year. The fact of physics, however, is that the feature size of a microchip cannot become zero. He adjusted his prediction shortly after by claiming every two year cycle, and later CPU manufacturers circumvented the limitations by starting putting several processing units into one chip. Other ways to increase performance include putting multiple CPUs on one motherboard, multiple printed circuit boards in one computer, or a number of separate computers in a distributed cluster.

A distributed system is the most promising way because it can interconnect computers with different hardware, lowering the costs of maintenance and upgrade. Scalability is very important for successful operations. For the multi-processor systems load balancing is capable of generating additional increase in performance, hence lowering overall cost of equipment and software. With the growing scale of distributed systems load balancing becomes an essential issue. The virtualization of resources including CPU time, network access and data storage becomes ubiquitous. Virtual systems are a multi-billion dollar industry and a very important technology for military, air-space applications, and consumer service providers. Cloud computing is becoming a strong niche for IT companies of all sizes.

This dissertation proposes, among other things, an artificial intelligence approach for load balancing, and its analysis for distributed systems running augmented reality simulation tasks. Fuzzy logic paradigm is one of the few effective techniques for load balancing when tasks are extremely volatile and unpredictable. The quantitative comparison was done for fuzzy logic load balancing intended for use in video stream generation within prototype comprehensive simulator ISE (Initiative Software Earth), a project of Computer Engineering Framework (CEF). Besides 3D visualization and digital communication simulation for robotic swarm interactions, other applications of the proposed approach include signal acquisition and signal and bitmap encryption. The mechanisms of virtualization with entitlement control make the test system a prototype cloud computing grid. The set of the rules put into the inference engines of the proposed approach have been proven to be sufficient to achieve the desired homogeneity of the CPU loads and other relevant parameters, hence increase the overall system's performance.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Parallel computation magnifies the performance

Majority of the computationally-intense tasks are being done using various levels of parallelism. Each parallelism approach demonstrates the best results for different applications. Pipelining is an example of hardware-implemented parallelism; multithreading is effective for the software supporting it. This dissertation among other things proposes a fuzzy logic algorithm for dynamic load balancing during 3D video rendering for a simulated robotic vision in a distributed cluster of computers (Figure 1). The algorithm has been tested within a comprehensive virtual reality simulator ISE (Initiative Software Earth), a project of Computer Engineering Framework (CEF), and the results show better performance than other approaches including static balancing and gradient balancing.

Widely known estimation made in 1965 by Gordon E. Moore (Fawwaz T. Ulaby, 2009) predicts that the number of transistors in the minimum-cost CPU doubles every two years. It may have been a coincidence at first (and he initially guessed doubling every year), but for the last 40 years it became well accepted by the designers and strategy-makers, so Moore's Law became a self-fulfilling prophecy of computer industry. The outstanding issue, however, is that feature size of a transistor cannot become zero, and the power density of a microchip cannot continue to grow beyond its melting point.

There are promising developments in nano-technologies and quantum computing, but the main reason computers increase their 'operations per second' ratio nowadays is parallelism. Dual- and quad-core CPUs are becoming ubiquitous; more CPUs are put on the motherboards and many video cards contain arrays of Graphic Processing Units.

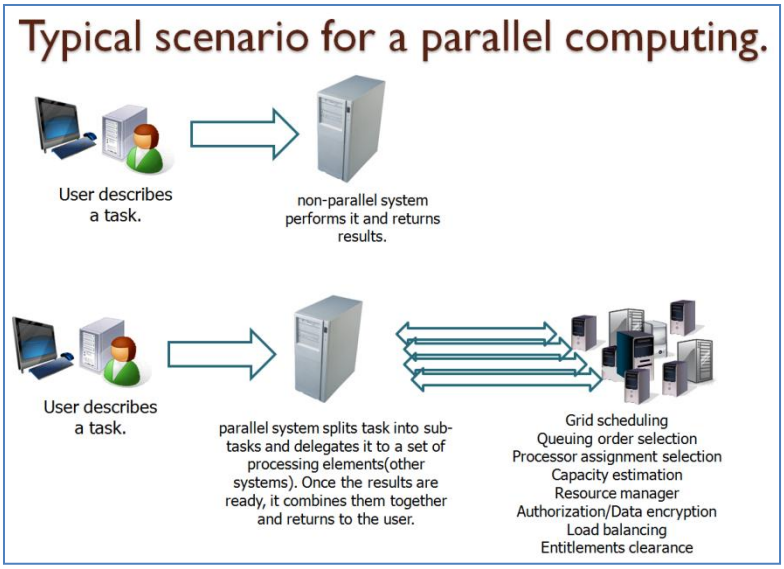


Figure 1. Parallel computing requires control over additional parameters.

One of the most powerful solutions is to combine computer architecture with effective network, that is, instead of attempting to incorporate as many CPUs as possible into one computer, a number of independent computers are connected together via a network switch (Figure 3). NASA's information power grid concept, together with the commercial cloud computing, will make high-performance computing power accessible to general users as easily as electricity from an electrical power grid (Leinberger, et al., 1999)(NASA, 2010)

The application level parallelism requires significant programming efforts. Often commercial or open-source frameworks used to run the distributed system (Boost),(GNU, Free Software Foundation), (Mathworks), etc. This eliminates the necessity to implement many intermediate layers among the servers. However, it deprives the researcher from full control over the system, and establishes an overhead for practically any interaction in the network. This overhead might be significant in case of contextual resource negotiation-based task allocation, even though some algorithms were proposed for complex multi-agent software systems (Jiang, et al., 2009).

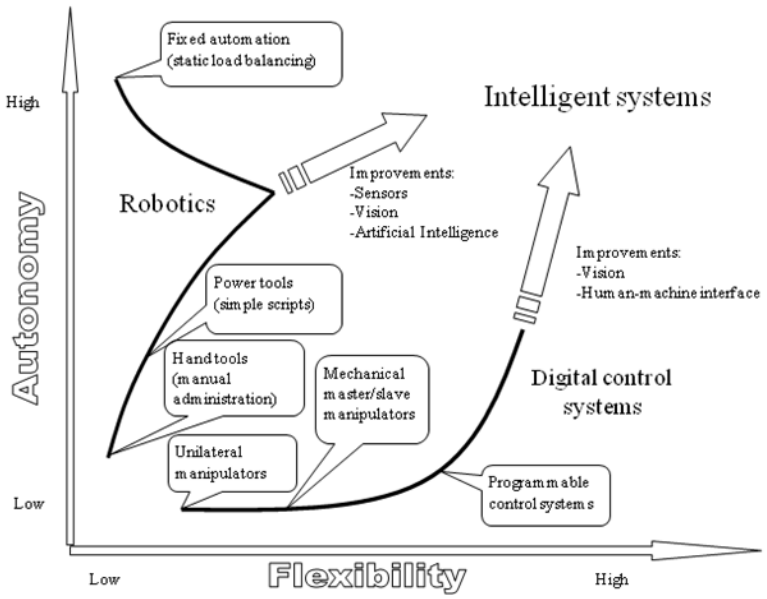


Figure 2. Merging of control systems and robotics into intelligent systems.

Load balancing could be viewed as an optimization of total time T_{tot}:

$$T_{tot} = \sum_{\forall Tasks} \left(\sum_{\forall Cycles} \left(t_{comp} + t_{comm} + t_{in\_repart} \right) + T_{DM} + T_{repart} \right) \qquad \textbf{(1)}$$

where t_{comp} is computation time, t_{comm} - communication time, t_{in_repart} - repartitioning within a cycle of the task, T_{DM} - time required for data migration (it can be optimized by effective caching), T_{repart} - is the repartitioning time for an individual task.

In general case, the optimization of the partitioning of the workload is hard to formalize *in details* because of the significant number of undetermined parameters. Even the CPU utilization for a processing element (PE) for a given task might not be known in advance if the CPU is used by several independent tasks; in addition, the execution of code is subjected to control by operating system (fluctuating processes' priorities). Although OS has the ultimate authority over the processes and threads, the distributed system can establish policies and try to

govern the execution on the application level, using some type of inter-process communication (Stevens, 1999).

Fuzzy logic as a case of AI

These considerations led to the realization that soft computing (a consortium of AI tools like fuzzy logic, neural networks, and genetic algorithms) is the most appropriate technology for the establishing an effective course of actions within the distributed system, in order to create highest degree of scalability (Abbot, et al., 2010). This course of actions of a distributed system might be extended beyond what is commonly referred to as 'load balancing', but in this thesis we address the repetitive changes within the cycle in (1). Hence the objective is to find better t_{in_repart} from the above formula.

Fuzzy logic and control systems were researched for a variety of different applications; Figure 2 shows the general directions of the evolution of robotics and digital control systems. These two areas are merging into intelligent systems (R.C.Dorf, 2008), which absorb all the most desirable qualities.

There have been many publications separately on load balancing, fuzzy logic, and 3D graphics. There are some papers on application of fuzzy logic to load balancing, and fuzzy logic to 3D graphics, for example, on level-of-detail management. This thesis presents novel approach that is on the junction of these three areas. This dissertation proposes an algorithm for dynamic load balancing in a distributed environment that has as its main workload a 3D video stream generation. The approach shows an improvement in the load balancing over other methods. The quantitative analysis has been done for two other algorithms and comparison with the existing state-of-art has been performed.

Target system architecture

There is distinct difference between architecture and implementation (Nguyen, et al., 2007). Implementation of the proposed approach and algorithms is NOT the focus of this thesis; instead, it is the architecture of a target system that needs illustration. And it is to illustrate the target system, I am introducing the simulation environment Initiative Software Ether (ISE) - a fully functional distributed system that was used as the testbed for the quantitative analysis.

ISE was implemented on the base of software package currently referred to as Computer Engineering Framework (CEF). CEF was started by me in 1989 to provide software engineering community with world class open source computing suit, its original design was focusing on 3D rendering and later offered a testing environment for image and signal processing algorithms (including signal acquisition for GPS). The term 'framework' was adopted in accordance with the contemporary trends in software engineering community.

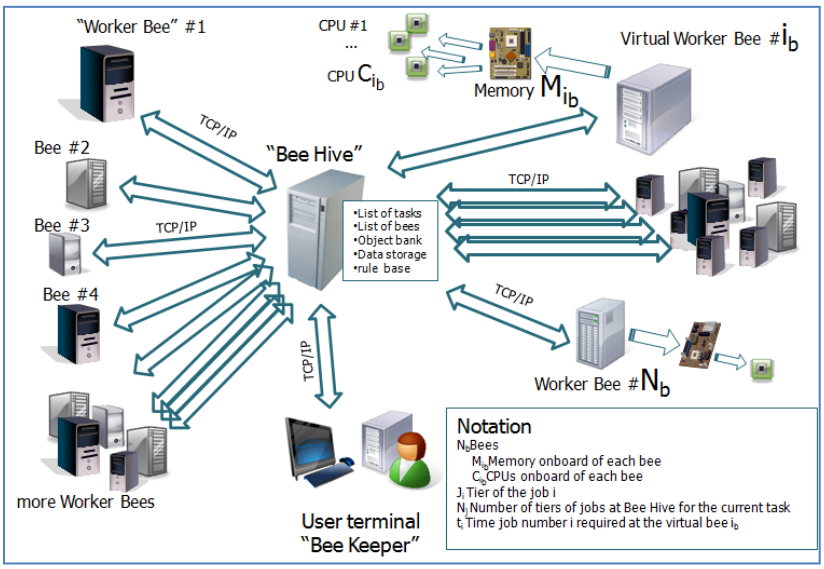


Figure 3. Topology of a distributed system.

Commercial entities, government and research institutions acknowledge the necessity to design, develop and maintain proprietary and/or open libraries that implement at least partially

the algorithms that are described in specifications. It is true for practically every area of industry or science. For example, the ones that have relevance to GPS cooperative positioning include C++ projects GPSTk(GPSTk), generic source codes for scientific programming could be found in the libraries of Free Software Foundation GNU (GNU, Free Software Foundation), Boost(Boost), SourceForge(SourceForge), and scripting integrated development environments (IDE) MATLAB(Mathworks), Simulink(Mathworks), LabVIEW(National Instruments), SPW, Octave(an open source alternative to MATLAB), SPICE (SPICE), also Symbian IDE(Symbian) and others.

Organization of the dissertation

The rest of this dissertation is organized as follows. Chapter 2 reviews the issues of load balancing for supercomputers and distributed systems; it overviews the state of the art systems. Chapter 3 reviews fuzzy controller design in general and current trends in state-of-art developments. Chapter 4 briefly describes the issues of video stream generation and 3D rendering technologies that represent the task - the subject of the load balancing. In Chapter 5 the proposed algorithm is formulated. Chapter 6 describes some of the details of the target system and the test system used to gather the data for the quantitative analysis. Chapter 7 shows the numerical results of the test runs. Section 8 outlines the opportunities for further research and practical applications of the proposed approach. Chapter 9 describes the programming tools, linkers and compilers used for implementing the proposed approach, as well as the platforms used, or considered to be used for the distributed environment deployment. Conclusion summarizes the thesis and states the further extension of the research.

CHAPTER 2. LOAD BALANCING FOR SUPERCOMPUTERS AND DISTRIBUTED SYSTEMS

Types of load balancing

Parallelism, and hence load balancing is an essential part of any contemporary computer system, be that a multi-core CPU or multi-node server farm. Depending on the task the system was designed for, the load balancing approach varies. The feasible algorithms for load balancing for a small-diameter multiprocessor system might be inefficient and hard to apply for big scale web-servers. Depending on the layer at which the load balancing is implemented, it might be done by a hardware block (Control Unit of a computer, multi-layer network switch, etc.), or a software program or sub-system.

In literature load balancing mechanisms are classified into two major types: static and dynamic. Static load balancing represents a scheduling mechanism to disseminate jobs among the participants, that is, among multiple CPUs, computers, network links, storage volumes, graphic cards, resident programs or other resources. Dynamic load balancing changes its policies or parameters depending on the work flow.

Depending on the nature of the participants, there are different targets for the optimization. Load balancing might pursue one or several of the following goals:

- maximize performance (e.g. multi-CPU systems)
- optimize resource utilization (e.g. storage frameworks)
- minimize response time (e.g. for network servers)
- increase reliability (via redundancy)
- bringing more profits to a web-represented business(via data mining)

Often some of the combinations of these goals are hard to achieve. For example, redundancy and optimal resource utilization are likely to increase server response time. Naturally, since most balancing algorithms can be described in the framework of optimization theory, it enables to involve classical results linked with convergence, its speed and other elements. The analysis of the load balancing based on the steepest descent algorithm and its speed of convergence can be found, for example in (Bronevich, et al., 2008). One of the classical dynamic load balancing methods is based on "gradient model", when the pressure gradient is defined by the requests from the idle processors. Under this pressure gradient, the backlogged tasks are transferred to the nearby idle processors. The publication (Lin, et al.) formulates the gradient model as to be independent of system topology, fully distributed and asynchronous. Global balance is achieved by successive refinements of many localized balances.

In some instances static load balancing is preferable to the dynamic. The variation of a static load balancing is when there is no exchanging of any information beyond the initial configuration of the system participants. For example, suggested in RFC 3074 Dynamic Host Configuration (DHC) Load Balancing Algorithm enables multiple, cooperating servers to decide which one should service a client (Volz, et al., 2001).

Concurrent programming

There are many models of concurrent programming, and mathematical description of communicating sequential processes (Hoare, 2004). Multi-process and multi-thread approaches are often combined and even utilized in energy-scalable embedded multiprocessor architectures (Schaumont, et al., 2005).

When the application is performing a task related to the video stream, there are certain aspects that need to be taken into account. First of all, common representation of a video stream

is block structure. Each block contains fixed or variable number of frames. Each of these frames, in turn, could be represented by blocks of images. In general case there is no guarantee in homogeneity of the computational complexity of either frames or portions of one frame. That is, some portions of a movie might be more computationally intense, or particular portions of a frame might take longer to compress/decompress.

The situation is more complicated when the video stream generation is being done using a number of nodes. If these nodes are not dedicated to a single task, their performance might be hard to predict. For example, if a node used for ray tracing, but in addition to that allows execution of digital signal processing application, there might be periods when the CPUs are extremely busy with one task or the other. If this node is expected to deliver a video frame, or a compressed sequence of frames, but slows down the delivery, the video stream generation might be delayed up to the point when the memory is exhausted and the entire task is halted to complete stop.

Hypergraph of Intelligent Agents

When dealing with local grid resource scheduling, the term 'hypergraph' usually refers to graph partitioning. In this case, vertices represent the computational load associated with data and the edges (hyperedges) represent data dependencies. This is the type of parallelism that was developed to increase performance of mesh methods, adaptive scientific computations, fluid dynamics, etc. Hypergraph manipulation methods have been and remain a very active research area (Catalyurec, et al., 2009), (Catalyurek, et al., 2007).

In this thesis, however, we are targeting bigger scale of scheduling. We are not referring to pipelining within a processing unit, but rather to global net of intelligent agents. Even though the agent processes may still be deployed within the same physical computer and use the same

CPU. To emphasize the hierarchy that may take place, and possible encapsulation, we are referring to this multi-agent system as a hypergraph.

Figure 4 shows the comparison of OSI, TCP/IP and CEF, our implementation, protocol architectures. The same intelligent agent can presume different roles in network. An agent can be an access point to a subnet of other agents.

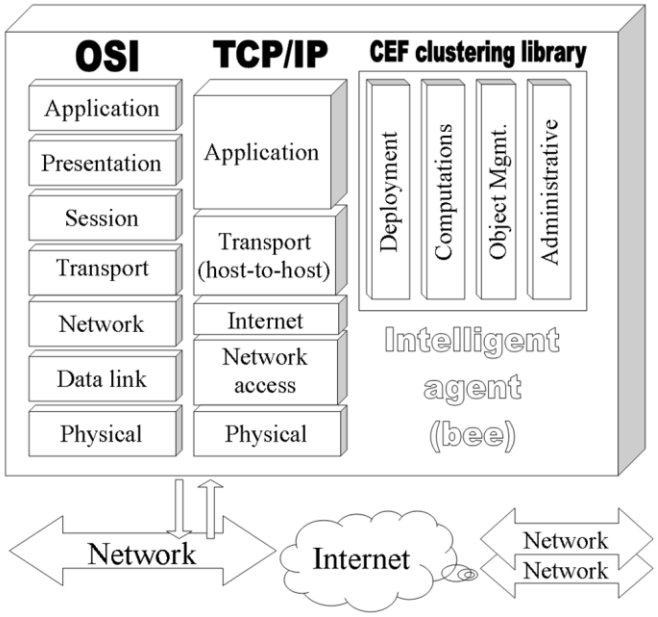


Figure 4. The proposed intelligent agent's layer structure with respect to the OSI and TCP/IP models.

CHAPTER 3. FUZZY LOGIC CONTROLLER DESIGN FOR A COMPUTER SYSTEM

Design of Fuzzy Controllers

Fuzzy Logic Controllers (FLC) are more and more widely used in recent years because they represent a solution in cases when the analytic expressions are too complex, but the heuristic explanation is too hard to program. Fuzzy Classifiers are also being actively researched (Nurnberger, et al., August 2007),(Jamshidi, et al., 2003), (Jamshidi, 2007), their application to neurobiology makes cybernetic organisms a reality.

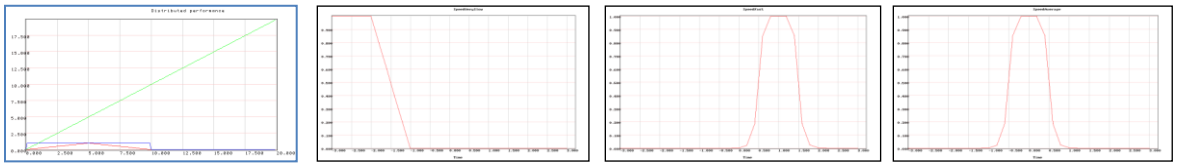


Figure 5. Examples of membership functions.

Fuzzy logic as a soft computing tool has many common concepts with statistics. However, historically it has developed its own terminology and is now placed by most researchers closer to practical artificial intelligence than to mathematics.

The basic item in Fuzzy logic is a Fuzzy variable, which is a function of one or more parameters. In discrete case, it is a set of numbers. And, by defining operations on these fuzzy objects, we can construct a consistent way of handling them as if they were regular numbers ('crisp' values).

It is common to give fuzzy variables intuitively self-explanatory linguistic tags. E.g. one can define a fuzzy parameter Temperature and assign values "hot" or "cold" to it, meaning that the value "cold" is some distribution in negative half of the temperature axis, and "hot" is on the positive side. The distribution functions that define fuzzy variables are referred to as

"membership functions" (MF).

Usually, membership functions are selected to comply with several common-sense rules: there should be at least one point in the MF's Universe (area where it is defined) that has value 1.0. Another practical rule requires that the left-most and right-most MFs had 1.0 up to the boundary. Figure 5 shows examples of membership functions. And Figure 6 demonstrates one of the implementations of fuzzy relationship "Y is much greater than X".

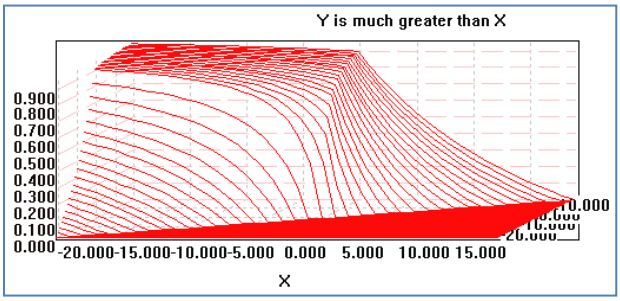


Figure 6. Example of fuzzy relationship "Y is much greater than X"

The common components of a fuzzy logic controller include five sequential blocks (Figure 7): Pre-processing, Fuzzification, Inference engine (contains the Rule base), Defuzzification, and Post-processing.

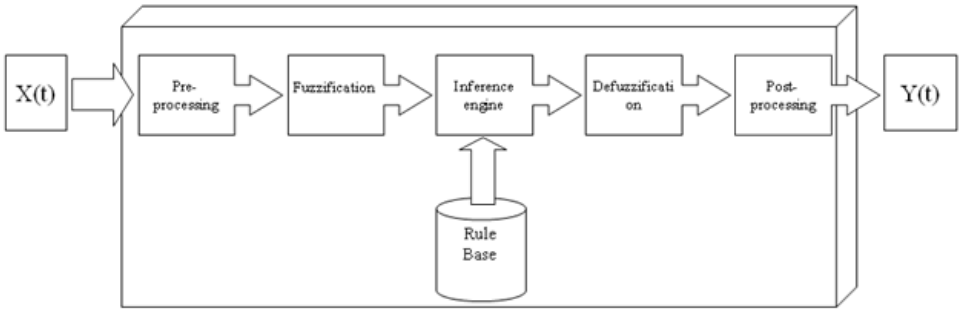


Figure 7. Blocks of a basic fuzzy logic controller.

Pre-processing adjusts the input parameters to a unified format. It could be scaling, bringing the data to certain sampling rate, filtering irrelevant interferences out, averaging, etc. For the tests we did we used non-linear normalization into the common range of values.

Fuzzification is procedure of converting the 'crisp' values into the fuzzy parameters. At this stage the fuzzification is applied to the input parameters. For example, we can fuzzify the execution time of the partition into "short time", "long time", etc.

Defuzzification is the inverse process, which is, analyzing the fuzzy output parameters of the inference block and determining the crisp data to be produced by the controller. Center of gravity, bisector of area (BOA), mean of maxima (MOM), or other approaches are commonly used.

Post-processing further refines the defuzzified values to the values appropriate for the actuator. For the load balancing tests performed, we had to have hard limitations on the partition sizes (it cannot be 0, and it cannot exceed the size of the whole job).

The inference engine is the main part of a fuzzy logic controller, and many approaches have been proposed. Rule base (heuristics) is programmed by a human and may contain linguistic definitions as well as fuzzy logic parameters. For example, boundaries and settings of the fuzzy logic membership functions, etc. The inference engine determines the way the rules from the rule base are executed. Their so-called firing strengths, in combination with their consequent functions, define the output fuzzy values. Examples of inference mechanisms include Mamdani and Takagi-Sugeno models (Nurnberger, et al., August 2007).

Control surfaces could be used to derive conclusions as to how to interpret the rule bases using as input the crisp values directly.

Big scale systems and OOP

In systems including significant number of interacting participants, the model outlined in Figure 7 can still be loosely used, but the complexity of the inference engine becomes hard to manage. For example, Figure 8 describes government income flowchart (R.C.Dorf, 2008). The

controlling mechanisms themselves are parts of the system; even though the diagram identifies one parameter (national income) the nonlinearity and non-time-invariance are substantial. If we tried to design an AI optimization of these interconnections, and use evolutionary algorithm, we would have to let the system itself generate the fitness function for the algorithm permutations.

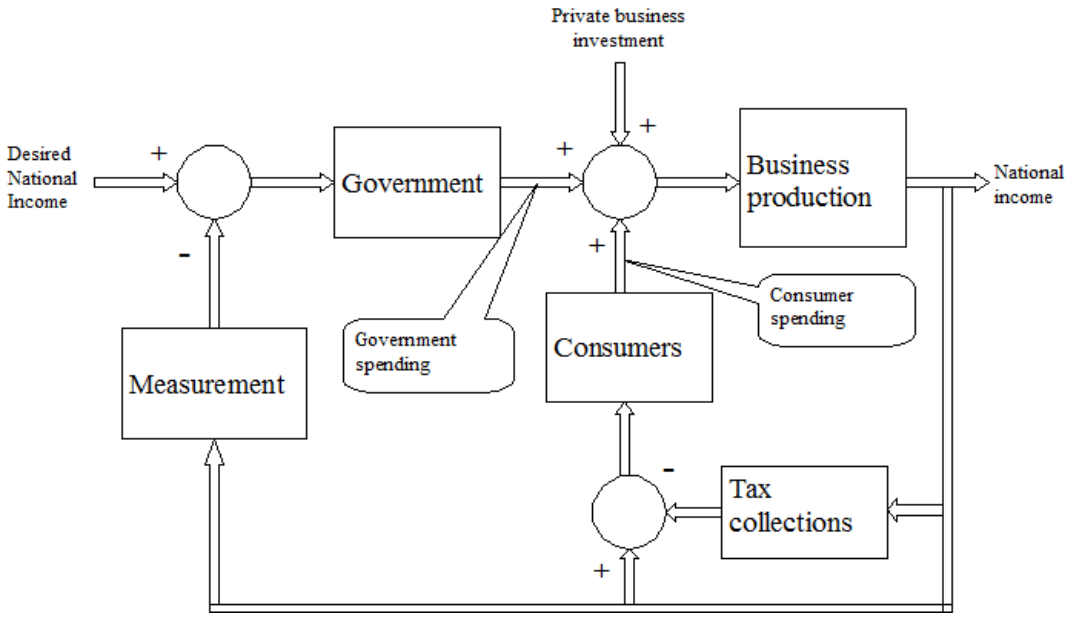


Figure 8. Example of a control system that is part of the system.

The reality is that blocks "Government" and block "Consumers" have a lot in common, in the sense of behavior or attributes; and the source of "private business investment" can be in fact a collection of individuals who are both Consumers and Government at the same time. This increasing complexity of the details of the model could be to certain degree resolved using the terms of object-oriented design.

CHAPTER 4. VIDEO STREAM GENERATION AND 3D RENDERING ISSUES

Video streaming

Since every load balancing approach has its strengths and weaknesses, depending on the set of tasks it is applied to, it is important to analyze the task and assure its adequate matching to the balancing control mechanism. This chapter gives brief definitions of 3D video rendering process, which is the subject of our workload partitioning algorithm that we describe in the next chapter.

3D rendering, and even to higher extent, video compression have become a good subject for algorithmic parallelism and effective hardware implementations (Purcell, et al., 2002).

Not only the video compression itself, but the computational challenges of the management of the consumer video delivery systems are a highly active area of research. Load balancing is important in video-on-demand service (Guo, et al., 2008)(Gerbec, et al., 2004), for cable television and especially for wireless broadcasting, which has relatively limited channel capacity.

Consumer video gaming industry is ready to have hardware ray-tracing for the high-end consoles. We were interested in the prototype 3D generation without using third-party hardware or software accelerators. The ray tracing, ray casting and other methods are notorious for the high computational costs, but very suitable for parallelism. This makes it an interesting subject for the custom load balancing.

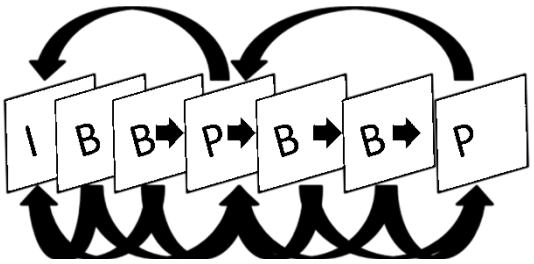


Figure 9. Example GOP sequence of MPEG encoder.

For every given frame, the task is to calculate the color components of every pixel in the bitmap. In case of uniform sampling rate, each instance of time reflected by a bitmap image. Physics engines and miscellaneous simulation models usually presume the time increments are uniform and monotonously grow. However, video compression for video streaming, during its execution, requests frames in non-sequential order. Figure 9, for example, shows the MPEG-encoding format sequence of intracoded (I), bidirectional (B) and predicted (P) frames within a Group Of Pictures (GOP).

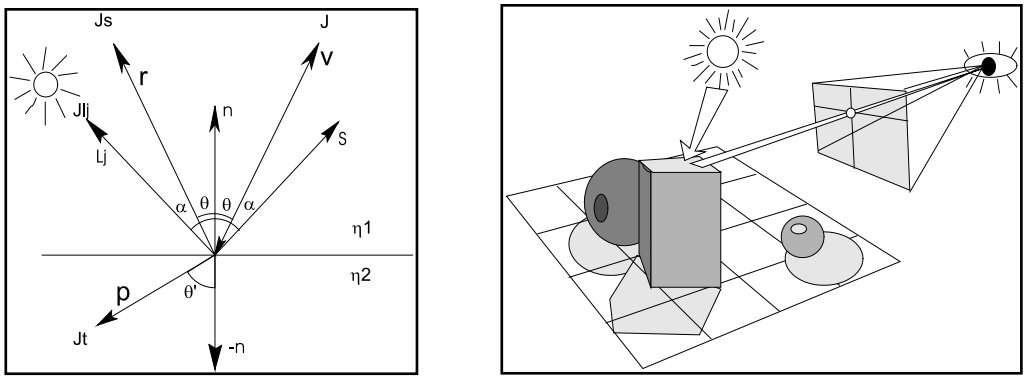
There are many reasons for choosing one sequence formula over another. For example, the bit stream optimization, i.e. by subsampling the chrominance values of the image, one can reduce the amount of the raw video data (Brown, 1995). For many practical reasons in the comprehensive virtual reality simulations it is preferable to have straight sequence. Indeed, if the modeling includes iterative solid body physics or hydrodynamics calculations, it is not feasible to skip frames and maintain big buffers with significant amount of data from the previous steps.

Commonly used models of light

Moreover, not only the sequence of frames can be skewed, but also the desired order of the pixels might be not orthogonal (left to right and top to bottom). E.g. zigzag scanning pattern of JPEG-based encoding affects negatively spacial homogeneity of the data as far as memory input/output operations concern. In this case hashing mechanism with insufficient memory will

have fewer hits, and more misses. This pattern aligns the video data for a video stream compression into required sequence.

In either case, each pixel's color components are defined by one (or multiple in case of casting or radiosity) ray tracing procedures. Figure 3 shows the diagram of the global model of light and the geometry of the classic ray tracing.



a) b)

Figure 10. The light a surface (left), image rendering.

The formulas of the model of light depend on the desired approximation of the parameters of the environment. Most of the tests we performed had the following expression for the intensity of the ray I_j :

$$I_j = [I_a K_a + \sum_{i=0}^{n} I_{l_i}(K_d \cos\theta_j + K_s \cos^{n_j}\alpha_i) + K_s I_s + K_t I_t]\frac{K}{d} \qquad (2)$$

,where

- I_a and K_a define the inherent illumination
- n - the number of sources of light
- I_{l_i} - the intensity of the visible light source l_i.
- K_d, K_s, and I_s - the reflective properties of the surface
- K_t, and I_t are the parameters of the refraction.
- K, and d - are the scale coefficient and the distance to the point of view.

For many applications this model produces sufficient visual realism and acceptable computational cost. In our case we applied it to the augmented reality project Initiative Software Earth (ISE) and were satisfied with the achieved level of realism. Of course, like with any 3D product, the artistic factor is important. For effective rendering, the shapes and the envelopes' hierarchy need to be designed properly. Starting from the early methods of ray-tracing, the 3D image generation required an effective geometry of the objects and their bounding volumes (Weighorst, et al., 1984). The shapes and hierarchy need to be designed well for effective rendering. Also, high geometric complexity could be replaced by carefully chosen texture bitmaps. This can simplify the scene, but significantly reduce the rendering time.

Figure 11. Examples of generated image.

Figure 11 and top image of Figure 13 show examples of video frames for the robotic vision simulation in artificial environment. It contains moderate amount of triangulated and non-triangulated shapes. To increase the workload, a number of glass-like objects have been introduced to stimulate ray stack bunching during refraction and reflection. Other scenes were also used for the tests, including indoor environment, space (Earth orbit), regular and irregular mazes. Practically any reasonable scene contains areas that are highly populated with objects, and areas that could be taken as an open space.

Virtual reality user interfaces and visualization have been a lucrative business for computer-aided design, architecture, consumer electronics, and entertainment. More advanced virtual reality systems offer precise modeling of the simulated environments, for example, contemporary flight simulators for airspace industry produce realistic simulation of the radio communication between the cockpit and the control tower, affected by noises and disruptions.

Comprehensive modeling of digital communication systems, as well as swarms of autonomous vehicles is complicated to implement in real time simulations. It requires big quantities of computational resources, CPU and memory. That was one of the motivational points for choosing video generation to be the subject of the distributed system. The load balancing is also suitable for operating with elaborate multi-tier models, involving cooperating robotic swarms and comprehensive digital communication simulations.

As many 3D animation products show, high geometric complexity could be replaced by carefully chosen texture bitmaps. This can simplify the scene, but significantly reduce the rendering time.

CHAPTER 5. TEMPORAL-SPACIAL ALGORITHM

Dynamic partitioning

Since every load balancing approach has its strengths and weaknesses, depending on the set of tasks it is applied to, it is important to analyze the task and assure its adequate matching to the balancing control mechanism.

The purpose of load balancing is to partition the amount of work that needs to be done among a number of processing elements (PE). In multi-core or multi-processor systems PE represents a CPU or a core; in multi-computer environment, PE also has separate memory and higher cost of interactions with the rest of the distributed architecture. Several main types of load balancing could be identified. Static partitioning is done before the computations started and carried out in uniform way until the completion.

Dynamic load balancing changes the partitions shapes and content during the execution; or changes the shapes of the graphs in case of hypergraph model(Catalyurek, et al., 2007). There have been developed libraries: Zoltan (Zoltan, 2010), PT-Scotch (Pelligrini, 2008), PaToH (Catalyurec, et al., 2009), ParMETIS (Karypis, et al., 2003), etc. Usually, call-back interfaces are invoked to use these libraries for variety of parallel computation applications. One of the most common applications of load balancing is network traffic scheduling(Shi, et al., 2005).

There are certain parameters of a task or restrictions of the chosen numerical methods that make some load balancing approaches work better. For example, for DSP applications or finite element methods the data locality might be important and data migration hard to implement; for some other applications only a very fast partitioner would make parallelism feasible.

Adaptive neural-network models, or combined fuzzy-neural network models have been

used for recursive nonlinear traffic prediction applied to MPEG video sources(Doulamis, et al., 2003),(Doulamis, et al., 2004). In our algorithm we use not the prediction of the compressed bit stream traffic, but the prediction of the complexity of the frame content.

The proposed fuzzy logic approach to dynamic load balancing is for a class of large-diameter multiprocessor systems. It is asynchronous; the model presumes the star system topology, which can be extended to star fractals with the introduction of higher level tasks, and made applicable to wide variety of cases.

Actor-based methods

Fuzzy visual systems could be automatically tuned using, for example, evolutionary algorithms (EA), for both single- or multi-objective cases(Muñoz-Salinas, et al., 2008). In part, the motivation for our proposed approach was development of a testing environment for robotic vision systems. That is, the design of an artificial environment that can generate consistent, reasonably realistic frames to be fed into the training sequences of the simulated individual robots' AI blocks.

The proposed approach is among the Agent- or Actor-based methods (Cao, et al., 2005). These methods are known to be well-performing in the environment of large number of processing elements with substantially non-homogeneous performances (CPU, memory, number of cores, network adapter's capacity, etc.), where the overhead due to the message passing is unpredictable and, in fact, not guaranteed.

Figure 12. Example of generated video frame.

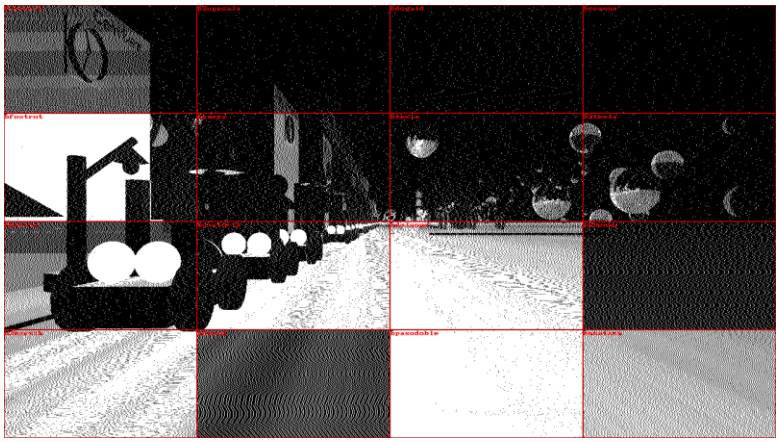


Figure 13. Example of the CPU time consumption pattern (in this case uniform rectangulars of 1/16th size).

The Actor model is based on object-oriented approach and states that the computational entities, in response to their input messages, can either send other messages or create other Actors. Actor model could be viewed as system of systems, since any encapsulated or encapsulating object is an Actor too.

When applied to the video stream generation, the Actors receive the messages containing the requests for the generation of areas of a frame bitmap, and send the response messages containing the generated array of pixel values.

The brightness of each pixel of the Figure 13 represents the CPU time required for

rendering. And each block's overall brightness represent each PE's resulting time. The higher brightness of a pixel indicates higher number of CPU cycles it took to perform the calculations. The total picture is the time required to complete a frame from Figure 12. It is worth mentioning that the particular implementation of the operating system functions that give the information about the CPU cycles, does not guarantee the precise values. Instead, it gives the number that is the best approximation. It is reflecting the fact that any performance monitoring imposes an overhead to the primary workload, hence, is undesirable.

With a sequence of frames put side by side, like on Figure 14, it is seen that the bitmap has the tendency to retain the brightness of certain areas. And it is not only because the areas have the same objects (same visual information), but also because of the nature of the 3D generation makes some areas more computationally expensive. A cluster of objects rotating on one place may cause significant changes in video stream, but keep its CPU cycle number relatively unchanged.

It is also important to take into account the fact that the complexity of a task is not necessarily the only factor determining the CPU usage. In low memory cases the operating system has to consume significant amount of CPU resources to satisfy the requested volumes of virtual memory. The use of virtual memory, in turn, affects, and is likely to surpass the actual time of the execution of the main workload because the input-output operations with hard-drives or network storage are in orders of magnitude slower than the regular memory fetching.

For this reason, and because of the nature of the concurrent Actor model, this thesis's approach suggests taking measurements of the actual time of the execution of a portion of the load.

Figure 15 shows the block diagram of the control code portion of the proposed algorithm. Each Actor A_i represents a Processing Element of the distributed system. A_i has, as its property, the record of the past jobs, $J(A_i, j)$. At the completion of each job, the record is updated and a new performance rating assigned to the A_i. The performance rating $P(A_i, j)$ determines how big a partition is assigned to the A_i during the next cycle.

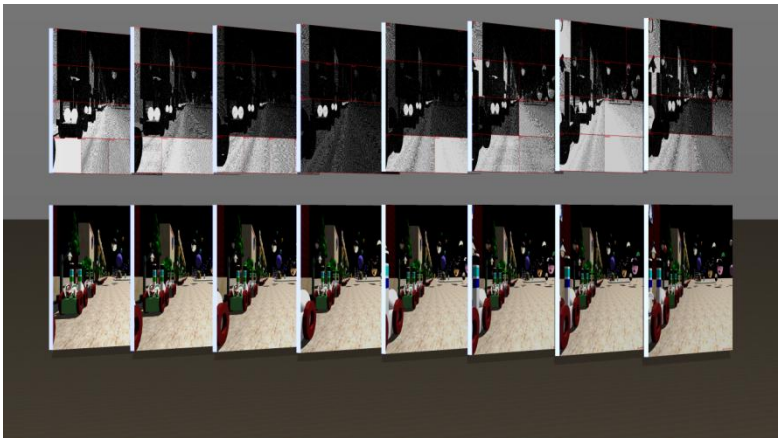


Figure 14. Set of frames (sequence on the bottom) and their time consumption with frame partitioning.

In general case, it is reasonable to partition the video-stream processing task into partitions that range from big groups of frames to small areas of the single frame's bitmap. The algorithm we propose is most effective for the frame-by-frame divisions. That is, each frame is divided into $X \times Y$ rectangular areas. Those areas are the atomic granularities of the load, and each partition is defined to contain between 1 and $X \cdot Y$ blocks.

As it was shown in Figure 13 and Figure 14, the time consumption snapshots are divided into 4×4 rectangles. Similar to the well-studied video compression observations, where some areas of the bitmap remain relatively unchanged during short periods of time, the time consumption of certain areas stays the same. As we pointed out earlier, this behavior is likely to

be true even in more cases than with the visual bitmap because of the locality of structures of the 3D objects and their envelopes.

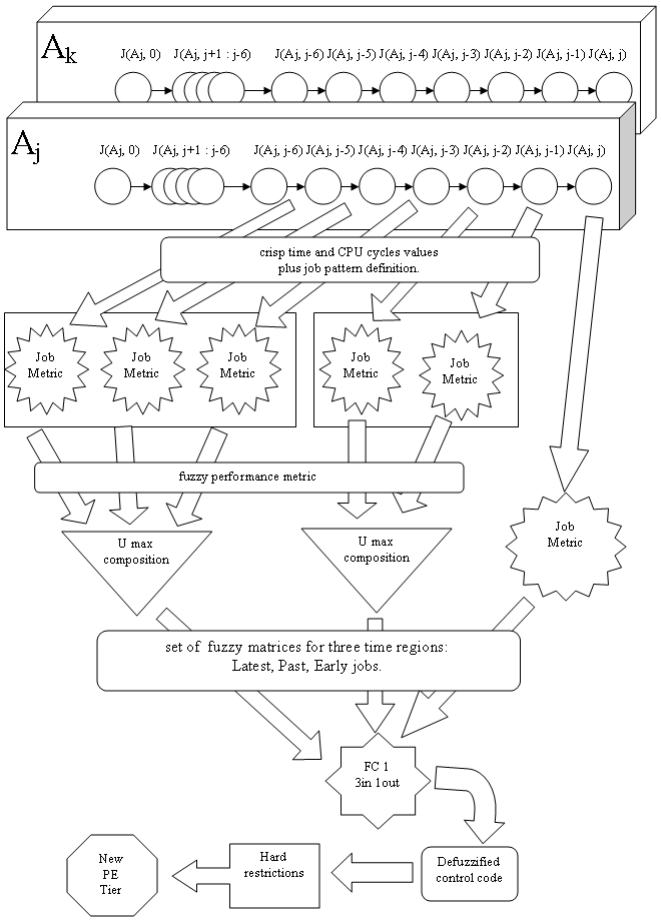


Figure 15. Block diagram of the control code portion of the algorithm.

In addition to the performance records of $J(A_i, j)$ in time domain, the controlling server is to collect the spacial records of the previous partitioning $S(t)$. $S(t)$ is a matrix $X \times Y$, containing the weight $S(t) = W(t, x, y)$ of the granularities of the bitmap. Note that because it is a spacial characteristics and the fact that the response time of the actors is not synchronized, at the time of the load balancing, some of the values in $W(t, x, y)$ may be undefined. We can presume that the weight $W(t, x, y)$ represents the computational cost of the granularity, but, of course, it is in fact the cumulative costs of computations and the message passing.

Start.
Initial stage:
- Receive a task (a 3D scene).
- Split the task into sub-tasks that have only one camera, continuous video stream.
- Split each video stream's frame into $X \times Y$ granularities.
- Initialize for $t = 0, W(0, x, y)$

Frame cycle stage:
- For the duration of the video sequence:

Start frame cycle:
- Pick available bee (connected and marked as idle).

If there is an issued (but not done) job for the Bee's Tier
Then
- assign this job to this bee and go to ***Start frame cycle***

Otherwise
- split the frame into granularities in such a way that there is a job with tier at least as low as the bee's current rating.
- Update the rating of the bee.

- Consolidate the done bitmap fragment
- Assemble the granularities into the frame and save it.
- go to ***Start frame cycle***.

Final stage:
- Save the statistics for individual bees to re-use in fuzzy logic evaluation of the processing elements.

End.

Figure 16. Pseudo-code for the block diagram part 1.

Not only $S(t)$ may be undefined for some indices, it also may carry discontinuities that were not the result of the video scene. It comes from the fact that because each $W(t, x, y)$ has been recorded from an actor $P(A_i, j)$, it is a part of some job, $J(A_i, j)$, and implies close coherence only within the mask $M(A_i, j)$ of that job. For example, Figure 13 shows that the jobs done by different PEs have rather different weights. In the middle, on the bottom there are two blocks with significantly different brightness. It would not have happen for the PEs with close performance parameters (CPU/memory/message-passing costs)

To address that, we propose to use performance normalization for different jobs according to the assigned performance rating of the PEs that did them.

The approach works well with non-sequential video stream generation. Because the

amount of the data, $W(t,x,y)$, needed for the evaluations is significantly less than the video frame bitmap itself. The fact that some frames will be rendered earlier is actually making the matrix $S(t)$ filled out with more data. In case of the predicted (P) frames, some estimation would have to be done, and the complexity sensing can be achieved using adjacent granularities.

Figure 16 summarizes the proposed algorithm.

CHAPTER 6. PROCESSING ELEMENT PERFORMANCE MEASURING

This chapter defines the job (partition) metric calculation of the proposed algorithm, and describes some details of the implementation we used to collect the presented test data.

There have been developed many types of fuzzy inference mechanisms. In the proposed implementation, as an intermediate inference engine, we used Mamdani approach. The implementation has two fuzzy logic controllers, they are marked FC_1, and FC_2 on the diagrams. The former has two inputs (antecedents) and one output (consequent). FC_2 has three inputs and one output. We used *Max()* composition for job metrics and the complexity estimation metric.

In cases of operating with the time parameters, the non-linear scaling is preferable for the fuzzification. And we used 5 linguistic tags for the jobs: 'Very Slow', 'Slow', 'Average', 'Fast', 'Very Fast'. The rule base for the FC_1, and FC_2 in general case should contain $5\times5\times5$ and 5×5 input tables. However, to make the FC faster, we also introduced fuzzy variables 'Any' and 'None'. It lowered the number of the parameters necessary to derive the decision during the load balancing.

The basic scenario programmed into the rule base includes rules similar to the following:

1: **if** (∃ *LatestJobs* & *LatestJobs* **are** DoneFast)
2: **and** (∃ *PreviousJobs* & *PreviousJobs* **are** DoneFast)
3: **and** (∃ *EarlyJobs* & *EarlyJobs* **are** DoneAny)
4: **then**
5: *PE* **is** Fast(AssignBiggerLoad)

The job metric takes into account the estimated time of the completion of this type of job for this type of area in this particular scene. It has the rules like:

1: **if** (∃ *ActualJobTime* & *ActualJobTime* **is** Short)
2: **and** (∃ *DesiredTime* & *DesiredTime* **is** Medium)
3: **then**
4: *Job* **is** DoneFasterThanExpected

We tried a variety of parameters and implementations of the membership functions. Smaller rule bases can work as effective as large ones, also, the controllers work faster if the fuzzy universe does not contain large amount of samples.

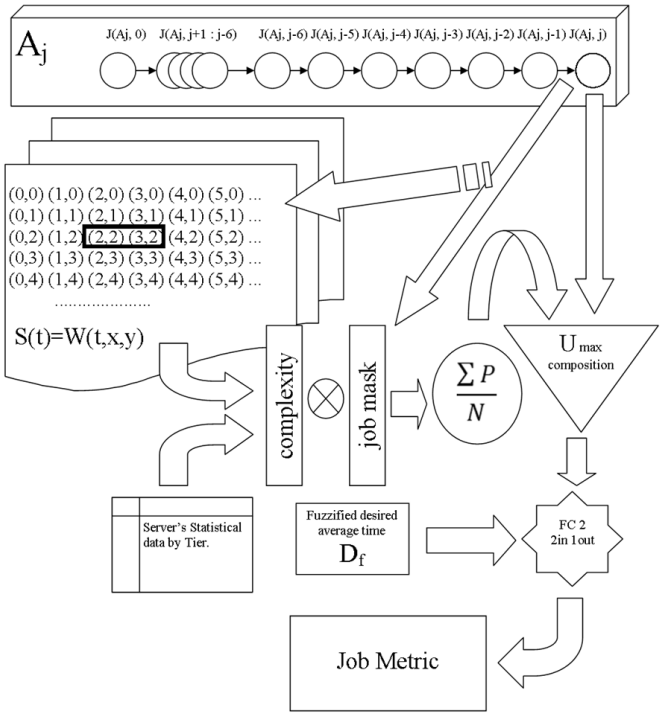


Figure 17. Block diagram of the job metric.

Large universes also occupy more memory per variable, which multiplies by the number of the processing elements and the number of the matrix $W(t,x,y)$ containing the past performance records.

For each of the controllers we did aggregation via max operator and centroid defuzzification. Other popular methods include middle of maximum, bisector, largest of maximum, etc.

CHAPTER 7. TEST RESULTS

Benchmarking of the performance of PEs

The proposed time-space optimizing algorithm was compared against others, including a static partitioning and a dynamic balancing without the region complexity sensing. In general case, to assess the computer cluster performance, it is useful to measure the benchmark of individual processing elements. There are a number of well known tests that are being run on different hardware platforms under different operating systems. They range from text processing (e.g. search/replace operations), to real-time network monitoring and scientific applications.

Num. of CPUs	640x480		1024x768	
	tracing(sec)	speedup	tracing(sec)	speedup
1	330	1.00	844	1.00
4	156	2.12	394	2.14
8	52	6.35	131	6.44
16	23	14.35	59	14.31
32	13	25.38	28	30.14

Figure 18. Speedup vs. resolution for parallel processing (from HiPEC, 1999)

It is impossible to design a computer system that would produce optimal results for any random application. Even if considering only the measurable parameters of a system, for some benchmark tests, a system can outperform others. For other tests the same system can show mediocre results. Non-measurable parameters include ease of use, maintenance and management. Good programming style and use of object-oriented languages like C++ facilitate the development process (Stroustrup, 2000), (Meyers, 1998). The effectiveness of business management is beyond the scope of this thesis; some material could be found, for example, in (Chrissis, et al., 2007).

In general, the architecture of a distributed system affects the test results. If a distributed grid contains distributed gateways that connect mainframe computers and the target servers, the use of the intermediate layer would be most efficient. In this case, according to (Shaout, et al., 1998), the fuzzy scheduling resulted in 16% improvement in the average total time to complete a job.

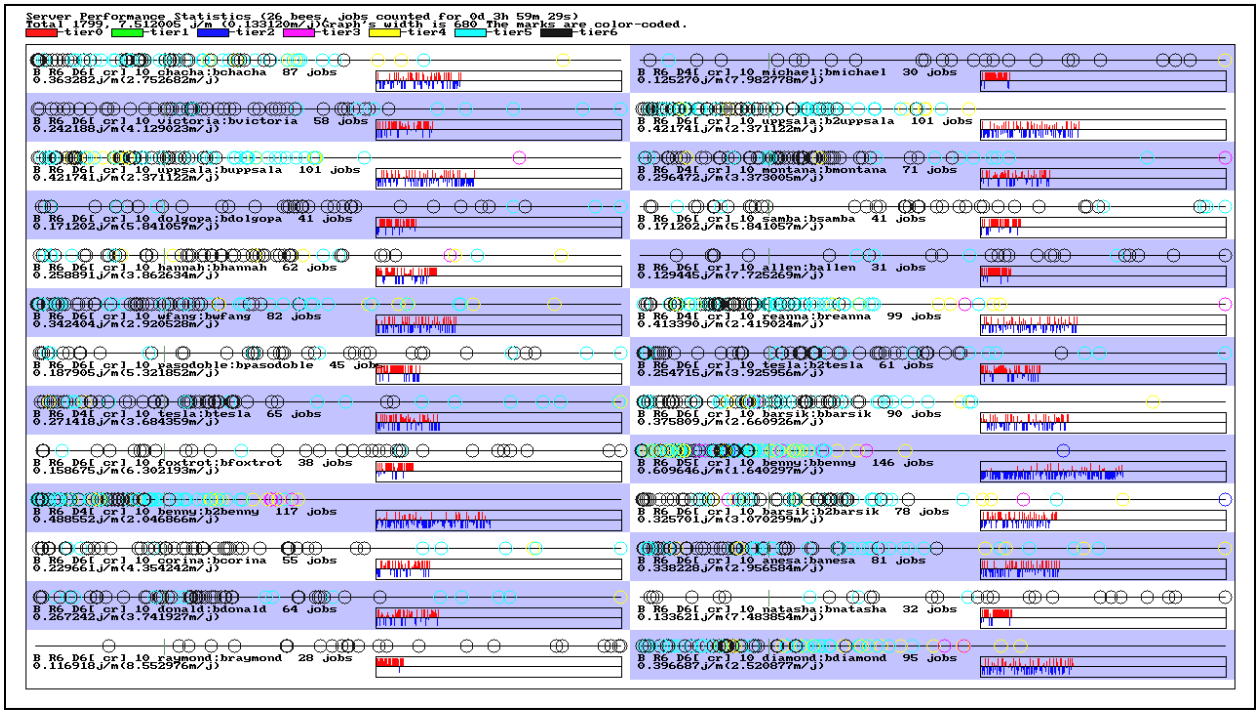


Figure 19. Example of the data monitoring window.

Fragment of the data of HiPEC, from (HiPEC, 1999), Figure 18 shows that the speedup is higher for higher resolution image generation. The effects of t_{comm} from formula (1) and other factors are also apparent. In comparison, CEF test runs were conducted on a system with CPUs with wide range of individual performances.

Test results for video generation

Figure 19 shows an example of the test run for a portion of the system. The one-dimensional scatter graphs consisting of the color-coded circles represent the individual job assignments for a particular PE (Processing Element). The bar graphs for each PE represent the time of the jobs with respect to the desired range of the completion.

Figure 20 shows the PE-assigned performance ratings, there is slight difference in the pattern of assigning numbers. The improvement is seen from Figure 21 - it shows that the better consistency, smaller variance and closer values to the target range. Just as expected, the test runs with the statically set parameters showed higher number of partitions which in particular implementation of the video stream generation (ISE simulator under CEF library) required additional re-compilations of the macrocode and additional computational overhead.

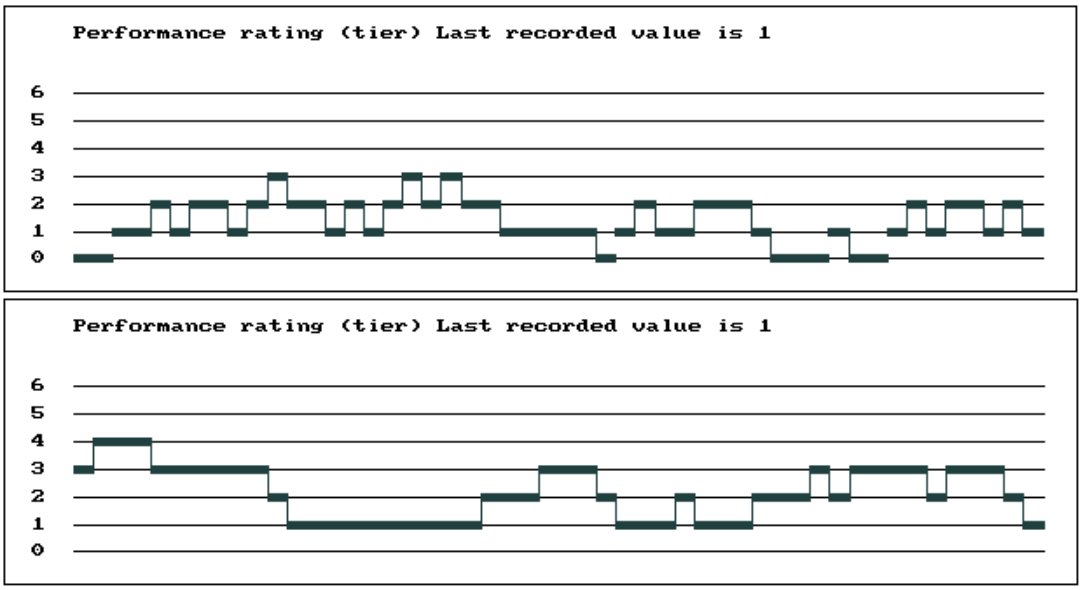


Figure 20.Tier assignment for Gradient (top) and Time-Space Optimizing test runs.

There were substantial differences in individual PEs' performances. As Figure 19 indicates, some of the CPU times were twice shorter. In addition to that, the topology of the

network included both 1Mbps and 100kbps network switches, meaning significant difference in message passing and the distributed object management handling.

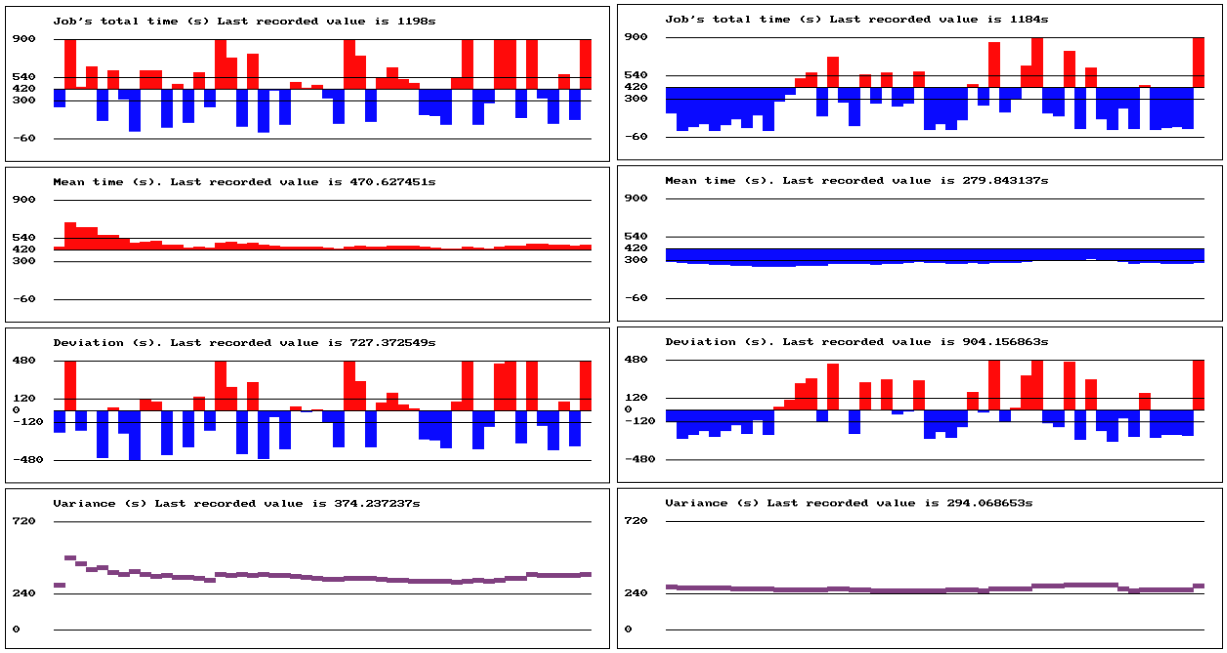


Figure 21. Job timing for Gradient (left column) and Time -Space Optimizing test runs.

The other test runs, for the scenes of different complexity, showed the time-space optimization work better for the scenes that have areas with significant relative computational cost. The graphs for calibration and comparison are shown in Figure 22, Figure 23, and Figure 24.

CHAPTER 8. OTHER PRACTICAL APPLICATIONS OF THE APPROACH

A novel scripting language YARC - script

Establishing effective load balancing in a distributed system allowed using its performance to the full extent. Artificial Intelligence, neuro-fuzzy and soft computing rely heavily on high-speed number-crunching computation (Jang, et al., 1997). Another element of AI has been researched and a new programming language has been designed. YARC script is a XML-like language describing object-oriented relationships among different entities, and their behavioral patterns. Despite the fact that it was intended originally for the 3D animation programming, it has all the capabilities of a scripting language with arrays, matrices, extended preprocessor, and a number of multi-dimensional data types.

Operators +,-,++,=, <, >, *, /, etc. are implemented and take operand of different types (numbers, strings, matrices, vectors or other objects). The list of operators is constantly growing and custom operator overloading paradigm was adopted in C++ language style.

Here is an example of YARC script's arithmetic instructions:

```
#define CHARGES_BASE
        [(0.001+SIN(time*4.0*2.0/USE_PERIOD)^2)*2.0*AREA_LENGTH, 0.0, 0.0]
#define FIELD_LAYER_NUMBER      0.5
#define THEPOINT      [         FIELD_ELEMENT_NUMBER*AREA_LENGTH_UNIT,        \
                                FIELD_LAYER_NUMBER*AREA_HEIGHT_UNIT,   \
                                FIELD_ROW_NUMBER*AREA_WIDTH_UNIT                ]
#define THEPOINT_FIELD1             (THEPOINT-(-CHARGES_BASE*0.5))
#define THEPOINT_FIELD1_ORT         (NORMAL(THEPOINT_FIELD1))
#define THEPOINT_FIELD1    (THEPOINT_FIELD1_ORT/(TV_Length(THEPOINT_FIELD1)^2))
        RotationAroundZ (       ((ComponentY(THEPOINT_FIELD_ORT) gt 0.0)
                        ? (ACOS(PROECTION_TO_XZ))
                                : (-ACOS(PROECTION_TO_XZ)) ))
```

- these are also demonstrating the pre-processor capabilities. And it is the way to do macro-programming, that is, using output of some scripts or other programs to feed into system as the human-written program.

And this is an example of Actor(in geometric sense) definition:

```
<ACTOR Version="2.2"; Name="ATAV_Radar"+ATAV_NAME;>
        <ParentName>ATAV_NAME</ParentName>
        <FormName>"ATAV_Radar"+ATAV_NAME</FormName>
        <MaterialName>ATAV_RADAR_MATERIAL</MaterialName>
        <ColorName>ATAV_RADAR_COLOR</ColorName>
        <ParentMat>TM_E      Shift [   -ATAV_LENGTH*0.45, //-ATAV_RADAR_HEIGHT*0.5,
                        ATAV_HEIGHT-ATAV_RADAR_HEIGHT*0.85, 0.0 ]
        </ParentMat>
        <InternalEnvelopeFormName>"NULL"</InternalEnvelopeFormName>
         <ExternalEnvelopeFormName>"NULL"</ExternalEnvelopeFormName>
        <SmoothingType>0</SmoothingType>
        <ReflectionType>ATAV_RADAR_REFLECTION</ReflectionType>
        <RefractionType>ATAV_RADAR_REFRACTION</RefractionType>
</ACTOR>
```

Establishing of the new language not only separated the user of the system from the platform-dependent programming, but also opened an opportunity for genetic programming using the operations and the operands as the nodes and the leaves of a formula.

YARC-script-defined motion pattern also can be used as elements of a set of chromosomes of evolutionary methods. This is one of the basic ideas behind the Petri dish concept of CEF robotic intelligence projects. Figure 28 shows a prototype robotic combat simulation.

Wireless network simulations

Another area of my research included comprehensive digital communication simulations. Here are some of the common characteristics expected from a state-of-art modeling tool:

- For the performance of a component, circuit or network to be adequately evaluated, there should be appropriate detail level of all factors which could influence it.
- The components should have common set of parameters so that simulated devices can be interconnected with each other to form compound devices, circuits or networks.
- Interfaces that pass sufficient information between the constituent components so that all possible interactions are identified.
- Computational efficiency that allows a tradeoff between speed and accuracy. This allows coarse or fine estimations of the system, which is especially important on the early stages of a project.
- Telecommunication and computer applications require the capability to simulate the components over a desired spectral range. This dictates the sampling rate and memory pre-allocation estimations.
- Digital communication research demands the ability to simulate advanced phenomena such as nonlinear effects, distortion in electromagnetic propagation, crosstalk, various models of noise and dispersion.

The contemporary computer applications often presume the presence of user-friendly way to control the execution flow. To enable a user to visualize and simulate a system quickly, the modeling and design programs usually have the following features (Keiser, 2000):

- The capability for the user to interact with the components during test run. The simulation may need to be adjusted or the combination of the parameters may need to be corrected. To make this possible at the early stages is critical for the efficiency of the simulation process. Early implementation of ISE application proposes fully interactive simulations sub-cycles that allow changing not only the test parameters, but also adding or removing components while the simulation run is under way (see Figure 25).
- The ability to create a system schematic based on the pre-defined libraries of the widgets or graphical icons. The graphical user interface (GUI) makes it easy to create various system components and the instrumentation (data sinks, spectrum analyzers, etc.) CEF proposes objects like CEFAnalyzer1D, miscellaneous attachments to the CEFAntenna device, as well as wide range of spectral analysis implementations. For example, there is an implementation of floating window FFT, it has the implementation of the generation of an image file for further data digging.
- The fundamental set of statistical-analysis, digital signal processing and display tools.
- Common visualization tools include eye diagrams and error-rate curves, the way to generate and describe the waveforms, CDMA sequences electrical and optical spectra, etc. (see Figure 31)

An advanced simulation tool at some point is required to have the level of control not achievable by standard buttons or configuration files. The best way to program the model is to use a programming language. Commercially available simulators often focus on so-called visual-

programming languages, such as LabVIEW(R), etc. In these languages, system components (transmitters, modulators, receivers and the connecting channels) are selected from a module library or programmed icons that have bidirectional or unidirectional interfaces (electrical, optical, etc.). Associated with each icon is a set of the operational parameters.

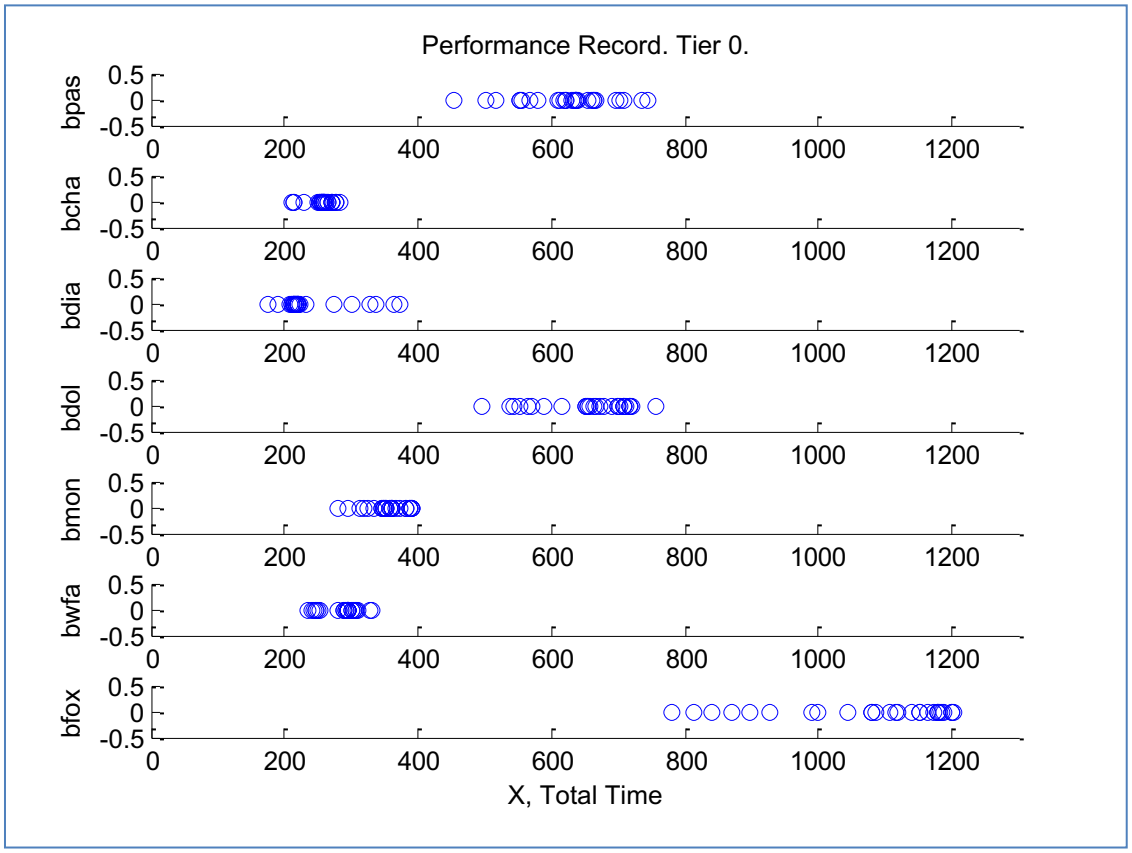


Figure 22. Performance evaluation of a number of nodes by the same tier jobs. (Y axis - job tier, 0 for everybody, X axis - total time per job)

Normally, a menu window is where the user specifies the values and the desired interface characteristics of a component. Sometimes in addition to the preprogrammed building blocks, users can create their own custom devices with either the underlying software code (a version of C or C++ for example) or the graphical-programming language. CEF has C++ as its underlying language and the mechanism to include the custom functionality extended to the ability of

creating completely new blocks or simulation entities. The proposed hierarchy of the objects allows extremely flexible integration of the modules.

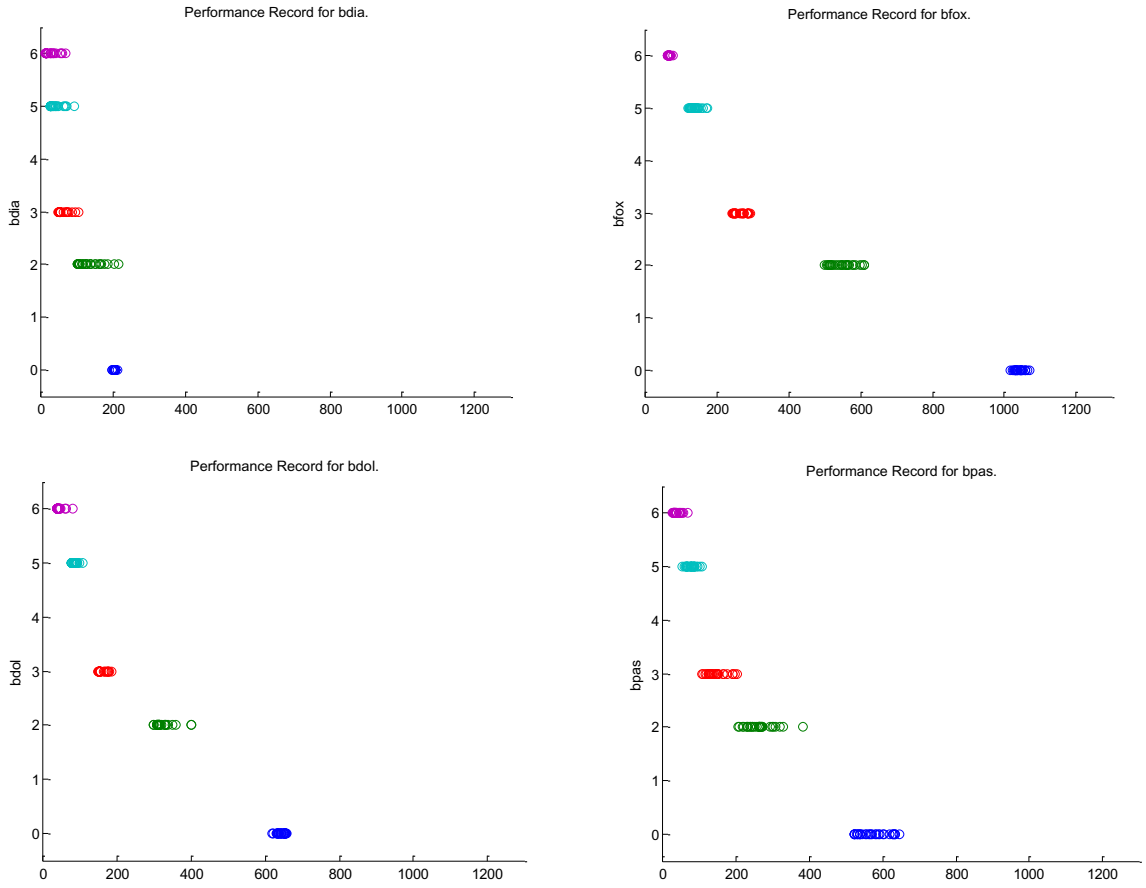


Figure 23. Performance of same node for various tier jobs. (Y axis - job tier (difficulty), X axis - total time per job)

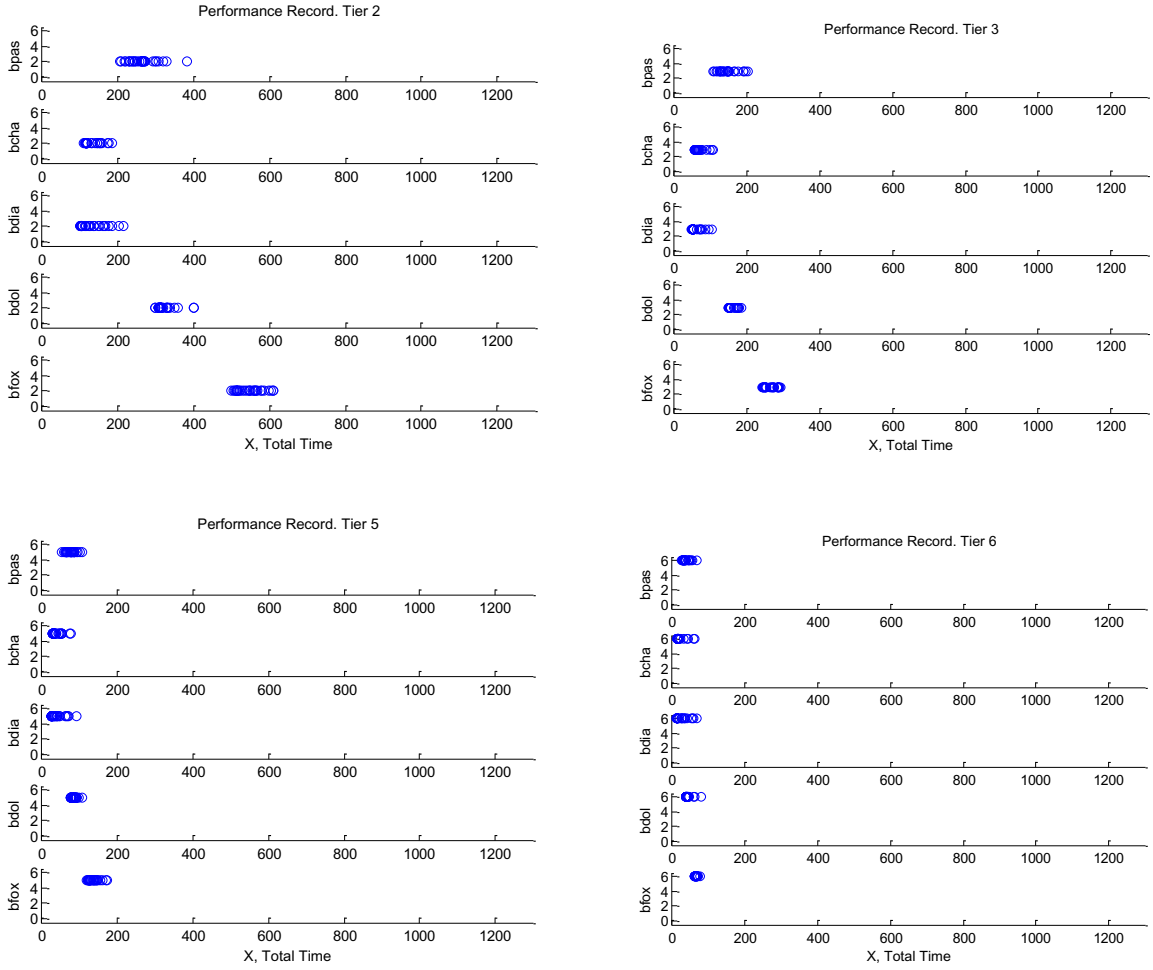


Figure 24. Performance evaluation for different tier jobs.

There are several fundamental definitions that define the basic structure of an object in the proposed ISE architecture. Most of the objects are inherited from C++ class CEFBase. Functionality dictates several layers of encapsulation. Once a component has been assigned some behavior, it is upgraded to a class CEFDevice, that has the application programming interface to methods like Process(). Those can be called at any other entry point and expect the parameters defined in the interface used. For example, a result of an algorithm applied to a one-dimensional signal, could be another one-dimensional signal class. So, the group of such devices is descended from class CEFDevice_1D_1D. The other combinations of the devices, including multiple input

parameters and multiple output parameters, have their own definitions and the API to assure the ability to interexchange one device for another.

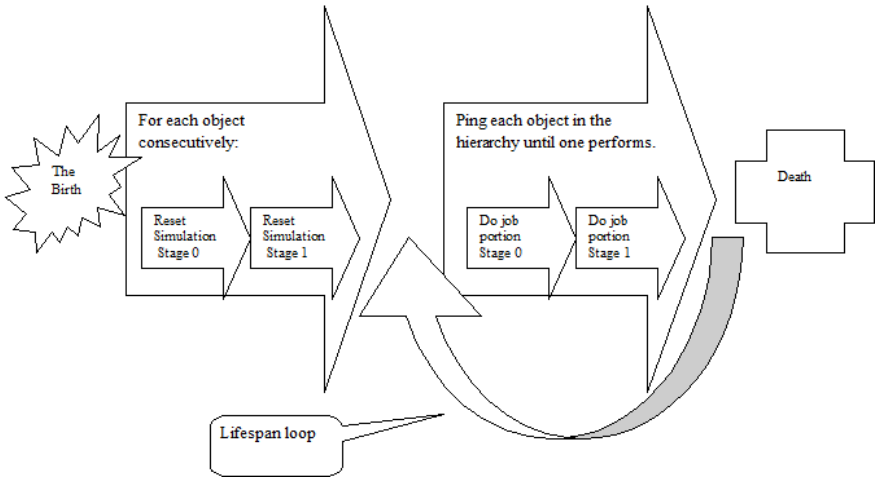


Figure 25. The lifecycle of digital communication simulator.

The life cycle of a root-level CEF ISE 3.0 object is shown on Figure 25. The proposed simulation sequence is described in term of the following definitions:

- Object's Birth - the time when the object is active.
- Object's Death - the time of detaching of the object from the parent object.
- Parent object - the object that has the current object as its component or parameter.
- Component - an object that performs some action. E.g. object Router has components Transmitter and Receiver.
- Parameter - an object that contains data. This could be Integer, Float, String, Image, Video, one- and two-dimensional signals, user-defined void pointer, etc.

Cooperative positioning simulation

Many corporations and research agencies test the feasibility of using a cooperative team of autonomous or remotely-controlled vehicles to perform tasks previously done by a human operator. In many applications, particularly military or including hazard environment, the choice of unmanned vehicles is strongly encouraged by government regulations. This tendency introduces new opportunities to the scientific community working on the positioning technologies. For example, as a part of a project for the Defense Advanced Research Projects Agency, Scandia National Laboratories' Intelligent Systems & Robotics Center developed a Differential Global Positioning System (DGPS) leapfrog capability that allows two or more vehicles to alternate sending DGPS corrections. Using this leapfrog technique, a group of autonomous vehicles can travel 22.68 kilometers with a root mean square positioning error of only 5 meters (JPEO, 2007).

Global Positioning System (GPS) as well as some other systems for determination of the device coordinates, is based on the time difference of arrival of beacon signals from precisely synchronized broadcasting stations with known location or pre-determined trajectory (satellites).

The primary sensor in the positioning device is an RF antenna that collects the data(Tsui, 2000). The positioning via calculations of the time delay is possible due to the finite speed of the message. In this case the receiving device can find the distance to the transmitting device. Figure 26 shows the general case, when both the beacon and the positioning receivers are moving. For the positioning applications it is important that the values of the perceived distances, called pseudo ranges, are related to the SVs that are not in the same plane. The precision of the user location in the perpendicular direction will be very inaccurate for the cases of beacons and the

receivers moving in the same direction. It demonstrates Doppler's effect very good, allowing analyzing the pseudo range accuracy.

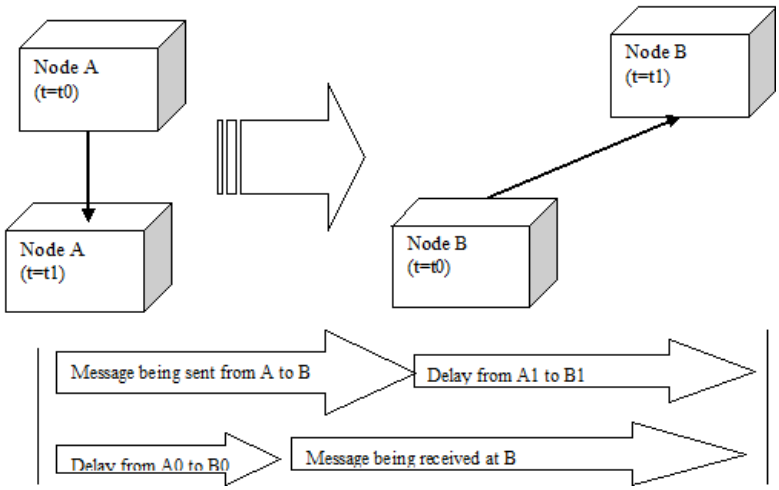


Figure 26. Basis for positioning using message-transmitting/receiving.

Acoustic or RF, cooperative positioning is a special case of digital communication simulations. GPS positioning are in effect a radio transmitting/receiving systems. With the current trend of all technology to shift from analog to the digital domain, the software radio could overcome many of the traditionally analog components. The open source community has its software radio branch. GNU Software Radio (FSF GNU, 2007) has many advantages, starting from the fact that it is an open architecture that eliminates royalties and license fees.

The network emulator can significantly improve the coverage and efficiency of testing, developing, and debugging cell phone software. Coverage is increased, since the network emulator can test fault and error conditions that are difficult or impossible to generate on real hardware (e.g. setting a particular level of QoS quality of service). It can also help analyze and test the services that have not been deployed in hardware, new versions of CDMA, Mobile IP, and next generations of mobile devices. (Harrison, 2004)

The proposed emulator, based on CEF library, at this point uses software interfacing to the devices (that may later be implemented in hardware). Some other emulators, for example, commercially available Symbian cell phone emulator, are mainly focused on the delivering hardware (new mobile devices) to the market. The Symbian network emulator has three components. First of them is The Emulation Environment, it is comprised of a set of PCs that emulate the external network. The purpose of those threads or processes is to function as the operator's network and attached data networks. This environment can emulate the behavior of the air-interface on data traffic, control services such as Mobile IP, PPP, TCP/IP or the others that run on top of them. The initiation and receiving of the connections has been implemented and can be monitored.

Cooperative positioning presumes that there are multiple participants on the field. That is why the uniform network node has been introduced. In the test system implementation, it is an autonomous vehicle that has all the components on-board. With well organized simulation environment it is relatively easy to replicate the instances of the Node class and have as many cooperating participants as the computer's memory allows.

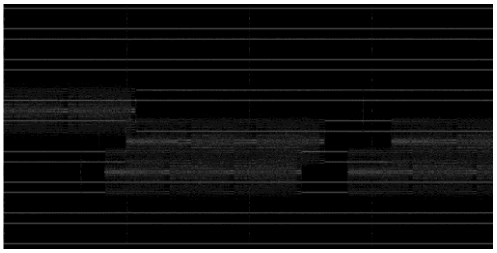


Figure 27. Wireless data transmission spectrum example

A simple layout with less than a dozen participants is shown on Figure 31. Some of the simulations have to have the obscure line of sight topography. In those cases it is more time consuming to generate the trajectories of the test subjects. One has to create and define the

timing, schedule all the turns and speeds. Example of a local test is presented on Figure 27, it shows an example of a spectrum snap shot of a simulation. The frequency hopping can be seen, as well as slight signatures of the boundaries of the synchronizing messages.

Computer-aided component design tools can offer a powerful way to assist in analyzing the design of a component, or a network before costly prototypes are built, or even instead of building a prototype. Important points that need to be considered are the approximations and modeling assumptions made in the software design. Most telecommunication systems are designed with several decibels of safety margin. That is why reasonably accurate approximations for calculating the operating behavior are not only acceptable, but also in most cases necessary to adhere to the feasible computation times.

Image recognition, evolutionary algorithms and sensor networks

This chapter briefly describes several concepts of artificial intelligence that were made possible with the introduction of powerful computers. They are made available to wider consumer groups with feasible implementations of distributed computation frameworks. Fuzzy logic is itself a tool for other AI techniques and algorithms. The areas of evolutionary algorithms, neural networks and artificial intelligence in general are the ones where parallel computations increase overall performance to unprecedented levels. Some of the internationally known projects connect hundreds and thousands of computers to process one single task. SETI, currently the largest distributed computing effort, linked over 3 million users (SETI Project). Computer Engineering Framework (CEF) has project Rozum, which is working on the human-machine linguistic dialog. Evolutionary approach is an interesting tool for solving complicated tasks where human might not be present and critical decisions have to be made fast.

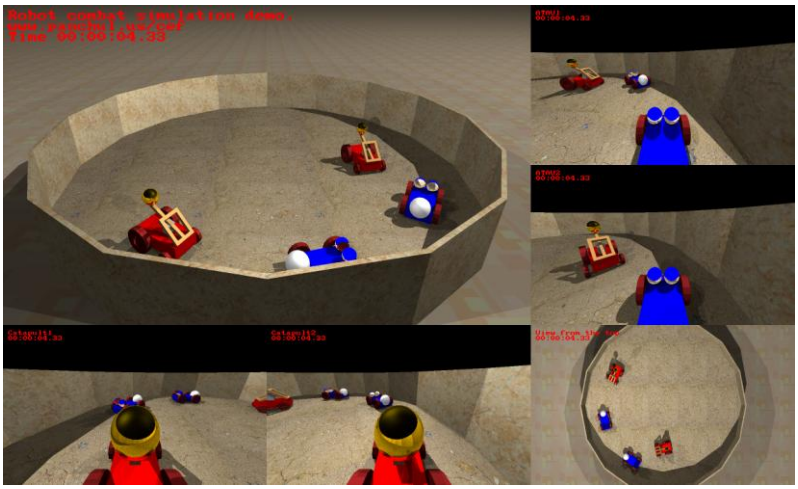


Figure 28. CEF-based virtual robot combat. Top view and each robot's views are shown.

Evolutionary algorithm approach offers stochastic, but directed, numerical search method for the best template of behavior of autonomous vehicles capable of information exchange among themselves. Evolutionary Algorithms and Genetic Algorithms in particular are known to be able to produce good results in tasks where regular approach is complicated or not feasible. The standard cycle includes the selection, crossover, mutations and reproduction.

Optical recognition is an essential part of an autonomous vehicle. Figure 29 shows the training set for the image identification of the bar-code vehicle tags. With the virtual autonomous vehicle moving in the simulated environment, it accumulates all the necessary angles and lighting conditions for the tests to be as close to the real deployment as possible. The codes in this example are 3 7-digit stripes painted over two cylindrical beacons, visible above the vehicle. Sometimes radio silence is preferable, so optical identification used.

Figure 29. Example of bar-code type of vehicle ID tags at different angles.

CHAPTER 9. IMPLEMENTATION PLATFORMS, LINKERS AND COMPILERS

Architecture and implementation are two different things

There is undisputed difference between architecture and implementation (Nguyen, et al., 2007). This dissertation introduces the proposed FL load balancing mechanism, and a number of aspects of the simulation environment this FL mechanism would be most effective in. Software embodies thoughts, ideas, design, implementation; each of which is an intellectual (and creative, not mechanical) process. Software engineering could be put even on another evolutionary level from other intellectual processes (like composing music or painting) because beyond a certain level of size and complexity, it is also necessarily a team activity (Morris, 2007).

To explain the architecture better, some implementation details are worth mentioning - according to CMMI models (Chrissis, et al., 2007), the process of implementation could bring positive feedback to the original design and the theoretical research.

Figure 30. The test grid, side by side there are around half of a hundred CPU blocks connected via network switches.

The consensus of majority of software producing entities is that the block structure is the preferable strategy for effective software development. Regardless of the size of a team of collaborators, or a number of contributors to an open-source project, the functionally self-

contained parts are often separated into reusable code, sometimes referred to as "toolkits", "toolboxes"(Panchul, et al., 17-21 April 2006), or even "tinkertoys"(Devine, et al., 2005). It is best accomplished when adequate design is done from the very beginning of project.

About software development.

Different versions of UNIX operating system were used. Some code was written for Sun OS, HP-UX, significant amount of user-interface utilities was written for Windows, and most of the system is on Linux RedHat 7.2, 9.0, Fedora, and SuSE 11.1 (64 bit).

A few words have to be said about the development environment in general. Beyond certain level of complexity, designing software is a big logistical task. It requires maintaining well-defined procedures of the source code version control. The introduction of new structural and algorithmic modifications should be done after thorough analysis as to which portions of the system are being affected and how.

In academic world it is very difficult because most of the programmers are not professional and keeping track of their modifications is not always possible. More often than not, when a graduate or undergraduate student joins a development team and starts writing some block of a system, he or she implements the functionality in the "quick and dirty" way, violating many rules of good style and focusing just on making things work. For small projects it may be acceptable, but for bigger projects, this means leaving a time-bomb for many future modifications. Disorder in naming conventions, static variables and dynamic memory usage, as well as the logical inconsistencies, might produce a number of critical, hard-to-catch, errors that will puzzle many collaborators.

The learning curve of the newcomers to software project in academia is often cut short because the students graduate and may leave more loose ends than robust re-usable code.

The safe haven for non-professional software development is the open-source community. For an open source project to survive, it has to have or acquire highly motivated group of collaborators, the core development team that oversees the evolving of the code, and controls the modifications and distributions.

In professional software development things are relatively easier to organize. Many people in industry came to accept the usefulness of the formally identified development cycles and at this point most of the big publicly traded companies received maturity level certifications (Chrissis, et al., 2007).

Computer Engineering Framework (CEF) is the project that, at this point, has already underwent several re-designs and absorbed both professional software development experience and educational one. Portions of the project were written to facilitate exercises or demonstrations in programming techniques, white others are in the prototype stage. The strategic plan is to keep the tools and implementations platform-independent and scalable. (Abbot, et al., 2010)

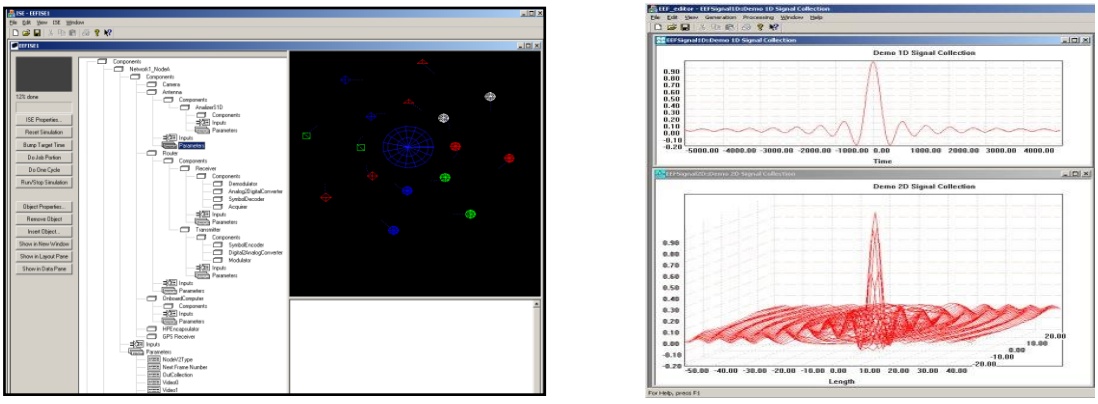


Figure 31. Overview of the tree of objects(left) and MFC views for signals(right).

Visual C++ .NET

As mentioned ago, user interfaces were often oriented on Windows operating system. Visual C++ .NET was used as the Integrated Development Environment (Microsoft Inc.). And it's basic GUI structure MFC (Figure 31).

In recent years, the virtualization of the cloud computer nodes has been moved to the hardware of the personal computers and servers. As with some RAID implementations, Microsoft is teaming up with the vendors to make the virtualization more effective.

Symbian 60 SDK, Carbide C++

We researched Symbian smartphones as possible platform for implementation of some of the techniques. Indeed, smartphones are potential processing elements (PEs) within a distributed cluster. In this chapter I briefly mention some of the tools for software development on Symbian OS. Some plug-ins for MS Visual C++ were used, and later, Carbide C++.

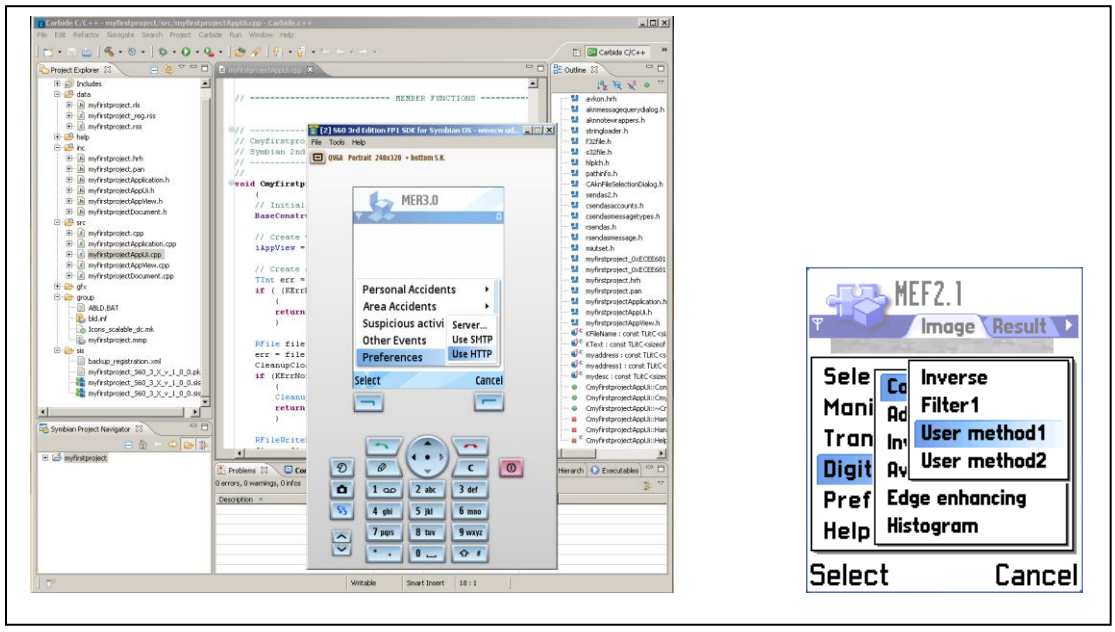


Figure 32. Carbide C++ IDE. Integration of a user method into MER 3.0

Symbian OS architecture influenced the evolution of CEF library. Remarkably, it is not a coincidence that the embedded software for cell phones eventually became a fully functional operating system. This fact validates the point of view that any big system, when its design is driven by the functionality, sooner or later transforms into a clone of UNIX (in a sense of having file systems, inter-process communications, etc.)

Overview of CEF structure

The software development is big area on its own, but when it comes to the engineering applications, every commercial or academic entity faces a choice of either writing proprietary programs from ground up, or using already available commercial packages. For the high level state of art implementations it is often preferable to write code from scratch. It requires more organizational efforts, manpower and money, but it sometimes delivers higher rewards too(Panchul, et al., 2010).

For any software package, it is important to develop a type-safe interface between the users of the package and its providers. This type-safety should not compromise the effectiveness of the library, that is, not to impose significant overheads in run-time or space on the user (comparing to the hand-written code) (Stroustrup, 2000).

It is well known that language shapes the way we think and determines what we can think about. Arguably, it is the object-oriented programming that makes artificial intelligence practical.

Figure 33 shows some of the blocks of CEF. The connections among them are omitted not only due to the lack of space, but also because these modules are building blocks of bigger systems. Computer Engineering Framework is growing. The term 'framework' is usually means a component that defines an abstract interface which provides a runtime loading and management mechanism for locating and loading plug-in component that provide concrete implementations

for its interface (Morris, 2007). In 'CEF' word 'framework' also has second meaning - collaboration among researchers, because making software (for any area, not necessarily numerical simulation) is not only an engineering process, but also a sociological or cultural phenomenon (McCarthy, 1995). That is, CEF does not have one narrow intended use, but rather it is a think tank for all existing and future technologies.

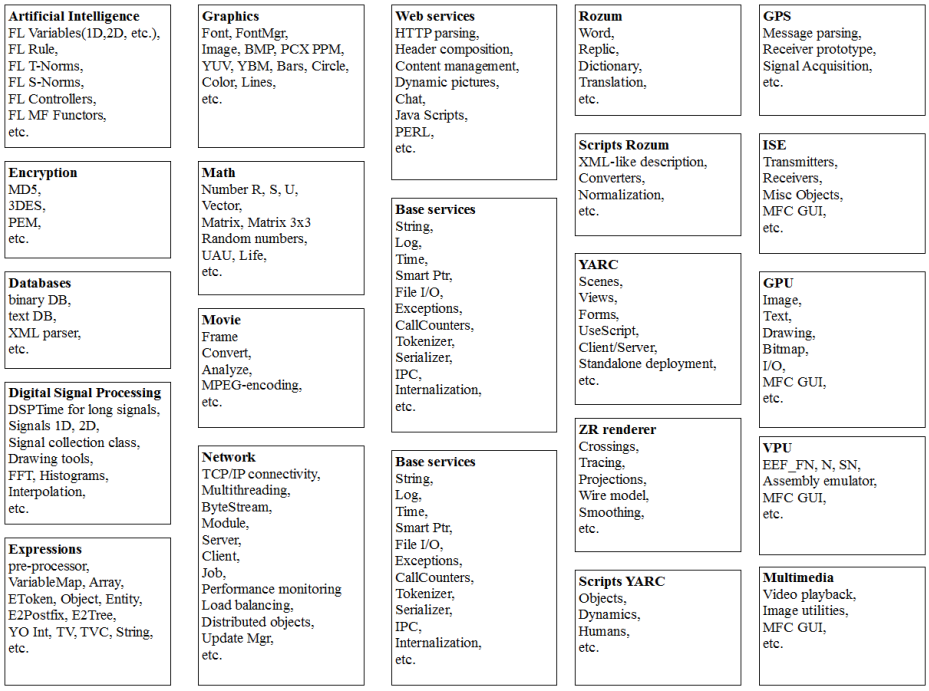


Figure 33. Partial block diagram of CEF.

Many of the blocks rely on each other and there are some basic objects that are used by most as parent classes.

C++'s support for design and programming has improved dramatically over the years(Stroustrup, 2000), and when it comes to efficiency of the code, C++ becomes the natural choice. On one hand, the very idea of classes, encapsulation, and polymorphism, is reminiscent of the way humans visualize the objects and their inter-relationships. This makes C++ suitable for high level programming. On the other hand, the roots of C++ are going to C language and,

just underneath C, Assembler. So, since C++ is a superset of C, it is also suitable for low level programming and fine-tuning of the machine code for a particular CPU or computer architecture.

ISE is a comprehensive simulation tool, intended to run tests of augmented reality technologies as well as a wide range of video processing. For example, algorithms for dynamic overlapping and non-overlapping camera networks (Junejo, et al., 2007), or three-dimensional image mosaicing using multiple projection planes (Chon, et al., 2007).

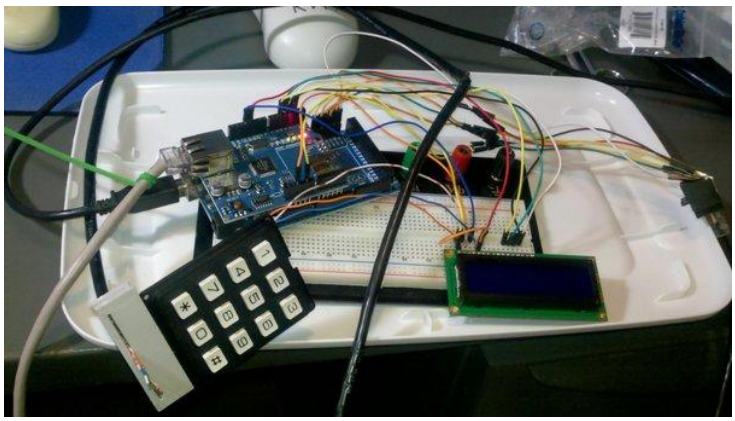


Figure 34. Prototype of Arduino microcontroller-based Ethernet robot.

The Internet-based deployment could be a convenient way to connect the sub-clusters. Figure 19 shows a generated server performance monitoring window that could be made visible to the cluster administrators via HTTP.

Hardware remarks

Implementing a prototype cloud computing cluster exposes many unexpected challenges. One of the ways to cut costs for the system as a whole is to cut costs of its components. The high-end computer hardware cost is growing or staying the same over time, while the low-end equipment is often available for free. The issue is to assess the ratio of performance per cost plus cost of maintenance. If a computer is producing very low operations/second, but still using same

amount of electricity, it has to be upgraded. Good example of low cost clustering is to combine the hardware parts of two computers to even out their performance, hence make them more suitable for parallel tasks. Another good example is network equipment. There are substantial advantages in custom crimping network cables.

Microcontrollers cannot compete with servers as far as performance is concern: the CPU and memory size are very different. However, in sensor networks, or swarms of autonomous robots, the pre-processing of data at the location would take some of the load from the system. In recent years microcontrollers and tools for their programming become more and more affordable. Figure 34 shows a proto-board with network-connected Mega2560, a version of Arduino board released with smaller board Uno. It features the Atmega2560, which has twice the memory, and uses the ATMega 8U2 for USB-to-serial communication microcontroller. The wireless add-ons and numerous open-source libraries are also available(Arduino).

CONCLUSION AND FURTHER RESEARCH

This dissertation proposed a concept for dynamic load balancing in distributed environments; with the focus on tasks related to video generation and virtual reality simulations. The fuzzy logic approach was proven to be effective when the constraints of the system are complicated and the load is not known in advance. The overall usage of Fuzzy logic increased the control over the functioning of the distributed cluster, and solved some of the issues related to the data migration and message passing.

The test runs showed improvement over other implemented methods. Among the measurable parameters, the variance of the individual job portions was decreased 78%, from 374 to 294. Other improvements were detected in smaller memory segmentation and shorter tasks backlogs.

The overall usage of Fuzzy logic increased the control over the functioning of the distributed cluster, and solved some of the issues related to the data migration and message passing. This enhancement opened the possibilities of commercial video rendering similar to HiPEC project (HiPEC, 1999), (Lüling, et al., 1998). CEF will continue creation of intelligent load balancing strategies for augmented reality comprehensive simulators. The possible expansion includes utilization of middleware implementations for distributed simulation applications (Wang, et al., 2009).

Future research includes further expansion of the algorithm to other aspects of digital simulations, further capitalizing on multi-processing and multi-threading, and possible hardware versions of the described techniques.

APPENDICES

Appendix A. An example of a source code for the proposed computer language

Below is an example of the proposed emulator microcontroller assembly language. The code calculates a number pi with any given precision.

Add up sum of the series for pi/4 = sum((-1)^(n+1)*1/(2*n-1))

The following is an initialization and the first iteration:

```
pushs 0
pushs 1
pushf                   ; initial sum, 0.0 pi/4 = sum( (-1)^(n+1)*1/(2*n-1)= sum(2/(16k^2-16k+3))
pushs 1                 ; current number
pushs 2                 ; ------------------------ NEW CYCLE
pushs 1                 ; denominator
pushf
pushs 3
pushs 1
pushf
dups                    ; preserve k
pushs 1
pushf
pushs -16
pushs 1
pushf                   ; -16 in F
mulf
addf                    ; -16k+3 in F
dups ; preserve k
pushs 1
pushf
dupf
mulf ; k^2
pushs 16
pushs 1
pushf
mulf                    ;
addf            ; 16k^2-16k+3 in F
divf                    ; 2/1 div 16k^2-16k+3 in F
addf                    ; sum is ready
pushs 1    ; incremented k
adds       ;
dupf       ; preserve sum
pushs 4
pushs 1
pushf
mulf       ; show 4*sum
clrl
showfieee 0
popf       ; we showed it alright
;
```

After 101 iterations the calculated number is:

3.1366419792...

- it is a very close approximation of 3.1415...

Appendix B. An example of object-oriented smartphone implementation

Computer Engineering Framework was made portable (Panchul, et al., April 2006). Its portions have been tested in the proposed Symbian-based system Mobile Electrical-engineering Framework (MEF).

MEF allows performing simulation in mobile environment or server-based collaborating network (Panchul, et al., September-October 2008). To facilitate the development process, there are several pre-designed templates that simplify programming. First of all, the algorithmic block needs an input and output. The input signal is the regular Image 1 tab (see Figure 32 for a screen shot of the application) content with all the properties and access application programming interface (API). The output of the algorithmic block is also a signal. Its dimension and properties are up to the algorithm itself.

There are a number of steps for attaching of the algorithmic block to the GUI structure of the application. The algorithm implementation a regular user is most likely to write will have a number of global variables or constants, maybe some auxiliary functions or initializing procedures. To hedge the system from all the problems, the algorithmic block is required to be derived from the CMEFProcessor class. It has abstract method Process():

virtual int Process(const CMEFSignal *inputsignal, CMEFSignal *Outputsignal)=0;

It is this method that after the instantiation is being called by the GUI class handling the events from the View. The minimalist definition of a custom algorithm implementation may look like the following:

```
#ifndef __COLOR_CUSTOM1_H__
#define __COLOR_CUSTOM1_H__
#include "MEFProcessor.h"
#include "MEFSignal.h"
class CMEF2AppUi;
class C2D_ColorCustom1 : public CMEFProcessor
{
```

```
public:
	C2D_ColorCustom1(CAknViewAppUi* myiUIptr);
	virtual ~C2D_ColorCustom1(void);
	virtual int Process(const CMEFSignal *inputsignal, CMEFSignal *outputsignal);
};
#endif
```

The internal variables and static data are added depending on the nature of the simulation. The processor class is made into the design structure called singleton. One instance of it is created and later re-used at every function call. In dealing with signal processing it saves significant time and resources, especially for the algorithms containing constant arrays with lengthy sequences that are being correlated during the block run.

The minimalist implementation of the Process() method looks as follows:

```
int C2D_ColorCustom1::Process(const CMEFSignal *inputsignal, CMEFSignal *outputsignal)
{
	if((!inputsignal)||(!outputsignal))
	{
		User::Leave(KErrGeneral);
	}
	if(inputsignal == outputsignal)
	{
		CMEFSignal *mytmpsignal;
		TInt myres;
		mytmpsignal = CMEFSignal::NewL(
	-------------Creating a copy-------------
				);
		mytmpsignal->AssignL(inputsignal);
		myres=Process(mytmpsignal,outputsignal);
		delete mytmpsignal;
		return myres;
	}
	// Check the aspect ratio if necessary
	if(inputsignal->NumberOfSamplesX		!= inputsignal->NumberOfSamplesY)
	{
		User::Leave(KErrGeneral);
//	e.g.	"Images are expected to be with the same width and height, equal to a power of 2."
	}
	outputsignal->InitL(
	-------- define the output signal -----------
);
//	MessageBox(NULL,"the function is under reconstruction!","",MB_OK);
	unsigned int thewholethingi=0;
	for(unsigned int j=0;j< inputsignal->NumberOfSamplesY;j++)
	for(unsigned int i=0;i< inputsignal->NumberOfSamplesX;i++)
	{
		outputsignal->SamplesRe[thewholethingi]=
		inputsignal->SamplesRe[thewholethingi]
+(thewholethingi%33);
```

```
                thewholethingi++;
            }
            outputsignal->AdjustMaxMinL(); //RecalculateMaxMin();
return 0;
}
```

There are several major parts that define the way a user interacts with an application: description of the menu hierarchy, internationalization database, handling of the user events.

The added algorithms are attached to appropriate menu commands that act on one or more tab images. The command should be added to the enumeration declaration of the commands being handled. It is done like so:

```
enum TMEF2CommandIds
   {
      EMEFHelp=1,
....
            EMEFRetouchColorCustom1_1,
};
```

Then the text string that is to appear at the menu item text should be defined in .loc file. For example:

```
#define qtn_mef_custom1_text       "User method1"
```

Eventually, the resource needs to be created:

```
RESOURCE MENU_PANE r_mef_retouch_color_submenu
  {
items ={...
     MENU_ITEM { command = EMEFRetouchColorCustom1_1; txt = qtn_mef_custom1_text; },
...      };
}
```

This command is handled in the base class of the View panels:

```
void CMEF2ViewBase::HandleCommandL(TInt aCommand)
```

Like so:

```
case EMEFRetouchColorCustom1_1:
   if(iSignal)
   {
      ((CMEF2AppUi*)AppUi())->GetInstance2DColorCustom1()->Process(iSignal, iSignal);
   }else
   {
      _LIT(mytxt,"Load the image first!");
      ((CMEF2AppUi*)AppUi())->OutputErrorNote(mytxt);
```

```
} break;
```

The instance of the custom1 algorithm is defined in the separate file.

The newly created class definition .h file need not be included into the project specification .mmp file of the application. Instead the source code .cpp file should be put via the following statement:

```
SOURCEPATH ..\src
SOURCE  ColorCustom1.cpp
```

There are several ways how to create the binary file and how to debug the obtained software using the command line or various IDE. Here are the steps to get all necessary components running using Microsoft Visual C++ .NET:

Open the command line window and go to the following directory: <SDK_installation_directory>\Series60Ex\MEF2\group\

Create the Microsoft Visual Studio .NET solution and project files: makmake mef2.mmp vc7. This will create the mef2.sln solution file and mef2.VCPROJ project file in the mef2\group\ directory.

Start the Microsoft Visual .NET IDE and open the solution file, mef2.sln, by selecting File > Open Solution from the menu bar. The Visual C++ IDE opens a new project window for the project, the MEF2 project window.

Build the application from the Microsoft Visual .NET IDE. Select Build > Build Solution from the Main menu bar (or press Ctrl+Shift+B). This will compile the source files and link the object files into a udeb build of the MEF2 application. The udeb build enables you to view the application in the emulator.

Start the emulator from the Microsoft Visual .NET IDE. Select Debug > Start from the Main menu bar. When starting the first time, the Executable For Debug Session dialog appears. Browse to the <SDK_installation_directory> and locate \epoc32\release\wins\udeb\epoc.exe.

There are additional steps as to how to install the application into the cell phone. The manufacturers of the phones provide this information. It could be done via Bluetooth, Internet or any other data transmitting technologies.

Appendix C. Abbreviations and Acronyms

Below are some of the abbreviations used in this Dissertation:

- AI Artificial Intelligence.
- ANFIS Adaptive Neuro-Fuzzy Inference System.
- API Application Program Interface.
- BOA Bisector Of Area.
- CE Computational Element.
- CEF Computer Engineering Framework.
- CF Core framework.
- COA Centroid Of Area.
- CORBA Common Object Request Broker Architecture.
- CPU Central Processing Unit.
- CTS Clear To Send.
- DGPS Differential Global Positioning System.
- DSP Digital Signal Processing.
- EDA Electronic Design Automation.
- EA Evolutionary Algorithm.
- EOM End Of Message.
- FFT Fast Fourier Transformation.
- FIS Fuzzy Inference System.
- FIFO First In First Out.
- FLC Fuzzy Logic Controller.
- FPGA Field Programmable Gate Array.
- FSF Free Software Foundation.
- GA Genetic Algorithm.
- GNU Recursive "**G**NU is **N**ot **U**NIX".
- GPP General Purpose Processor.
- GPS Global Positioning System.
- GPSTk GPS Tool Kit.
- GPU Graphic Processing Unit.
- GUI Graphical User Interface.
- GUI control an input field, check-box, scroll-box, tree-view, buttons, etc.
- HDL Hardware Definition Language.
- HTTP Hyper-Text Transmission Protocol.
- HW Hardware.
- ISE Initiative Software Earth.
- ICWG Interface Control Working Group.
- IDE Integrated Development Environment.
- IDL Interface Definition Language.
- IO Input/Output.
- IP Internet Protocol.

- JPEO Joint Program Executive Office.
- JTRS Joint Tactical Radio System.
- LabVIEW Laboratory Virtual Instrumentation Engineering Workbench (R) National Instruments Inc.
- LSB Least Significant Byte.
- MF Membership Function.
- MHAL Modem Hardware Abstraction Layer.
- MOM Mean Of Maxima.
- MSB Most Significant Byte.
- OCR Optical Character Recognition.
- OOP Object Oriented Programming.
- PE Processing Element (a node in a multicomputer or multiprocessor)
- PID Proportional-integral-derivative controller.
- QoS Quality of Service.
- PDA Photonic Design Automation.
- RF Radio Frequency.
- RPC Remote Procedure Call.
- RTS Request To Send.
- SA Simulated Annealing.
- SCA Software Communications Architecture.
- SETI Search for Extra-Terrestrial Intelligence.
- SOAP Simple Object Access Protocol.
- SPICE Simulation Program for Integrated Circuits Emphasis.
- SPST Single-Pole, Single-Throw (on/off switch)
- SPW Signal Processing Worksystem (R) CoWare Inc.
- SV Space Vehicle.
- SW Software.
- TCP Transmission Control Protocol.
- TSP Traveling Salesperson Problem.
- UML Unified Modeling Language.
- VHDL VHSIC Hardware Description Language.
- VHSIC Very High Speed Integrated Circuit.
- VPU Virtual Processing Unit.
- VCS Version Control System
- WF Waveform.
- XML Extensible Markup Language.

BIBLIOGRAPHY

Papers published based on the research presented.

Journal

- A.Panchul, D.Akopian, M.Jamshidi "Temporal-Spatial Fuzzy Logic Algorithm for Effective Dynamic Load Balancing in Distributed 3D Graphics Engine“ Accepted for publication in International Journal System of Systems Engineering

Conferences

- A.Panchul, D.Akopian, M.Jamshidi “On Object-Oriented Implementation of a Soft Computing Toolbox for CEF” World Automotion Congress(WAC), September 2010.
- A.Panchul, D.Akopian, P.Chen, "Development of Smartphone-Based Public Vigilance Systems. WACONG, Hawaii September 2008
- A.Panchul, D.Akopian, "Development of educational applications for smartphones“ 2008 ASEE Annual Conference & Exposition June 22-25 Pittsburgh, PA
- A.Panchul, D.Bhupathiraju, S.Agaian, D.Akopian, "An imaging toolbox for smartphone applications", Mobile Multimedia/Image Processing for Military and Security Applications, SPIE Defense and Security Symposium, 17-21 April 2006, Orlando, FL
- P.Sagiraju, A.Panchul, GVS Raju, D.Akopian, "Development of Software GPS+ Receiver Testbed," IEEE PLANS'2006 Conference, April 25-27, San Diego, CA.
- A.Panchul, D.Akopian, "Educational Applications Development for Smartphones" UTSA ESB '2007 Conference November 9, 2007
- A.Panchul, D.Akopian, "On porting computer applications into Symbian cell phone platform", IEEE Region 5 Conference, April 2006, San Antonio, TX.

REFERENCES

A.L.I.C.E. Project An Educational Software that teaches students computer programming in a 3D environment. [Online]. - 2010. - www.alice.org.

Abbot M. and Fisher M. The Art of Scalability: scalable web architecture, processes, and organizations for the modern enterprise [Book]. - [s.l.] : Pearson Education, 2010.

Apollo Photonics, Inc., Ontario, Canada [Online]. - 2010. - www.apollophoton.com.

Arduino Arduino Hardware [Online]. - 11 01, 2010. - http://arduino.cc.

Boost www.boost.org [Online]. - 2010.

Bronevich A. and Meyer W. Load balancing algorithms based on gradient methods and their analysis through algebraic graph theory [Journal]. - [s.l.] : Academic Press, Inc. Orlando, FL, USA, 2008. - Issue 2 (February 2008) : Vol. Volume 68.

Brown D.C. A Bitrate Control Algorithm for the Berkeley MPEG-1 Video Encoder. Report No. UCB/CSD-97-937 [Report]. - Berkeley : EECS University of California, 1995.

Cao J. [et al.] Grid Load Balancing Using Intelligent Agents [Journal] // Future Generation Computer Systems. - January 1, 2005. - No.1 : Vol. 21. - pp. 135-149.

Catalyurec U.V. and Aykanat C. PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0 [Report]. - Ankara, Turkey. : Bilkent University, Dept. Computer Engineering, 2009.

Catalyurek U.V. [et al.] Hypergraph-based Dynamic Load Balancing for Adaptive Scientific Computations [Conference] // Parallel and Distributed Processing Symposium. - [s.l.] : IPDPS 2007 IEEE International, 2007.

Chon J. [et al.] Three-Dimensional image mosaicking using multiple projection planes for 3-D visualization of roadside standing buildings. [Journal] // Transactions on Systems, Man, and Cybernetics.. - [s.l.] : IEEE, August 2007. - 4 : Vol. vol.37. - pp. 771-783.

Chrissis M., Konrad M. and Shrum S. CMMI, Guidelines for Process Integration and Product Improvement [Book]. - Boston, MA : Pearson Education, Inc., 2007.

Devine Karen and Hendrickson Bruce Tinkertoy Parallel Programming: A Case Study with Zoltan. [Journal]. - [s.l.] : International Journal of Computational Science and Engineering, 2005. - No 2/3/4 : Vol. Volume1.

Doulamis A.D., Doulamis N.D. and and Kollias S.D. An Adaptable Neural-Network Model for Recursive Nonlinear Traffic Prediction and Modeling of MPEG Video Sources. [Journal] // IEEE Transactions on Neural Networks. - 2003. - No.1 : Vol. 14.

Doulamis N.D. [et al.] A Combined Fuzzy-Neural Network Model for Non-Linear Prediction of 3-D Rendering Workload in Grid Computing. [Journal] // IEEE Transactions on Systems, Man, and Cybernetics. - April 2004. - No.2 : Vol. 34.

Fawwaz T. Ulaby Michel M.Maharbiz Circuits [Book]. - [s.l.] : National Technology and Science Press, 2009.

Feddema J., Lewis C. and LaFarge R. Cooperative Sentry Vehicles And Differential GPS Leapfrog [Article]. - 2000.

FSF GNU GNU Software Radio Documentation [Book]. - [s.l.] : FSF, 2007.

FSF GNU Project Software Radio. [Online]. - http://www.gnuradio.org.

Gerbec D. [et al.] Determining the load profiles of consumers based on fuzzy logic and probability neural networks [Journal]. - [s.l.] : IEE Proc.-Gener. Transm. Distrib., 2004. - 3, May : Vol. 151.

GNU, Free Software Foundation [Online]. - 2010. - www.gnu.org.

GPSTk GPSTk website [Online]. - 2009. - www.gpstk.org.

Guo J. [et al.] Combination Load Balancing for Video-on-Demand Systems [Journal]. - [s.l.] : IEEE Transactions on Circuits and Systems for Video Technology, 2008. - 7, July : Vol. 18.

Harrison R. Symbian OS C++ for mobile phones. Volume 2. Programming with Extended Functionality and Advanced Features. [Book]. - [s.l.] : John Wiley & Sons Ltd, 2004.

HiPEC HiPEC, High Performance Computing Visualisation System supporting Networked Electronic Commerce Applications. [Report] : Public Final Report of ESPRIT HPCN PST Activity. - [s.l.] : AXCENT GmbH Dr.Karsten Morisse, Technologiepark 13, D-33100 Paderborn, 1999.

Hoare C.A.R. Communicating Sequential Processes [Book]. - [s.l.] : Prentice Hall International (original 1985), 2004.

Jamshidi M. Large-Scale Systems: Modeling, Control, and Fuzzy Logic [Book]. - Englewood Cliffs, NJ : Prentice Hall, 2007.

Jamshidi M., Vadiee N. and Ross T. Fuzzy Logic and Control: Software and Hardware Applications. [Book]. - Englewood Cliffs, NJ : Prentice Hall, 2003.

Jang J and Sun C. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence. [Book]. - Upper Saddle River, NJ : Prentice-Hall, Inc., 1997.

Jiang Y. and Jiang J. Contextual Resource Negotiation-Based Task Allocation and Load Balancing in Complex Software Systems. [Journal]. - [s.l.] : IEEE Transactions on Parallel and Distributed Systems, 2009. - 5 : Vol. 20.

JPEO Joint Tactical Radio System Standard Device Packet Application Program Interface [Book]. - [s.l.] : JPEO (Joint Program Executive Office), 2007.

Junejo I., Cao X. and Foroosh H. Autoconfiguration of a dynamic nonoverlapping camera network. [Journal] // Transactions on Systems, Man, and Cybernetics.. - [s.l.] : IEEE, august 2007. - issue 4 : Vol. vol.37. - pp. 803-816.

Karypis G., Schloegel K. and Kumar V. ParMETIS: Parallel graph partitioning and sparse matrix ordering library, version 3.1. Technical report [Report]. - [s.l.] : Dept. Computer Science, University of Minnesota, 2003.

Keiser G. Optical fiber communications [Book]. - [s.l.] : McGraw-Hill, 2000.

Leinberger W. and Kumar V. Information Power Grid: The new frontier in parallel computing [Journal]. - [s.l.] : IEEE Concurrency, 1999.

Lin Frank C. H. and Keller Robert M. The Gradient Model Load Balancing Method. [Journal]. - [s.l.] : IEEE Press Piscataway, NJ, USA. - Issue 1 : Vol. IEEE Transactions on Software Engineering archive. Volume 13.

Lüling R. and Schmidt. O. HiPEC: High Performance Computing visualization system supporting networked electronic commerce application [Conference] // EURO-PAR'98 Parallel Processing. - Springer, LNCS 1470 : [s.n.], 1998.

Mathworks Matlab website [Online]. - 2010. - www.mathworks.com.

Mathworks Simulink [Online]. - 2010. - www.mathworks.com/products/simulink/.

McCarthy J. Dynamics of Software Development [Book]. - [s.l.] : Microsoft Press, 1995.

Meyers S. Effective C++, Second Edition [Book]. - [s.l.] : Addison Wesley Longman, Inc., 1998.

Microsoft Inc. [Online] // Microsoft website.. - 2010. - http://www.microsoft.com.

Morris Ben The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS [Book]. - [s.l.] : John Wiley & Sons, Ltd, 2007.

Muñoz-Salinas R. [et al.] Automatic Tuning of a Fuzzy Visual System Using Evolutionary Algorithms: Single-Objective Versus Multiobjective Approaches [Journal] // IEEE Transactions on Fuzzy Systems. - April 2008. - No.2 : Vol. 16.

NASA NASA Advanced Supercomputing [Online]. - 2010. - http://www.nas.nasa.gov/.

National Instruments National Instruments [Online]. - www.ni.com.

Nguyen H. [et al.] A Case Study OF Context-Driven Application As An Example Of Educational Project On Cell-Phone Platforms [Conference]. - San Antonio, TX : UTSA ESB '2007, 2007.

Nurnberger, A. Klose and Andreas On the Properties of Prototype-Based Fuzzy Classifiers [Journal]. - [s.l.] : IEEE Transactions of Systems, Man, and Cybernetics, August 2007. - number 4 : Vol. Volume 37 number 4..

Panchul A. [Online] // CEF website. - 2010. - www.panchul.us/cef.

Panchul A. [et al.] An imaging toolbox for smartphone applications [Conference]. - Orlando, FL : Mobile Multimedia/Image Processing for Military and Security Applications, SPIE Defense and Security Symposium, 17-21 April 2006.

Panchul A. and Akopian D. Development of educational packages for smartphones [Conference]. - Pittsburgh, PA. : ASEE Conference, 2008.

Panchul A. and Akopian D. Educational Applications Development for Smartphones [Conference]. - San Antonio, TX : UTSA ESB '2007 Conference, November 9, 2007.

Panchul A. and Akopian D. On porting computer applications into Symbian cell phone platform [Conference]. - San Antonio, TX : IEEE Region 5 Conference., April 2006.

Panchul A., Akopian D. and Chen C.L. Development of Symbian Smartphone Based Public Vigilance System [Conference]. - Waikoloa, Hawaii. : ISIAC 2008, September-October 2008.

Panchul A., Akopian D. and Jamshidi M. On object-oriented implementation of soft computing toolbox for CEF. [Conference] // World Automation Congress(WAC). - Kobe, Japan : [s.n.], 2010.

Pelligrini F. PT-SCOTCH 5.1 user's guide [Report]. - [s.l.] : Research rep., LaBRI, 2008.

Purcell T. [et al.] Ray Tracing on Programmable Graphics Hardware [Journal]. - [s.l.] : ACM Transactions on Graphics, 2002. - 3, pp 703-712 : Vol. 21.

R.C.Dorf R.H.Bishop Modern Control Systems [Book]. - Upper Saddle River, NJ : Pearson Education, 2008.

RSoft, Inc. RSoft, Inc., Ossining, New York [Online]. - www.rsoftinc.com.

Sagiraju P. [et al.] Development of Software GPS+ Receiver Testbed" [Conference]. - San Diego, CA : IEEE PLANS'2006 Conference, April 25-27 2006.

Schaumont P. [et al.] Cooperative Multithreading on Embedded Multiprocessor Architectures Enbales Energy-scalable Design. [Conference] // DAC. - Anaheim, CA, USA : ACM, 2005.

SETI Project Search for Extra-Terrestrial Intelligence [Online]. - www.seti.org.

Shaout and McAuliffe P. Job scheduling using fuzzy load balancing in distributed system [Journal] // Electronics Letters. - October 1st, 1998. - No.20 : Vol. 34. - pp. pp.1983-1985.

Shi W., MacGregor M. H. and and Gburzynski P. Load Balancing for Parallel Forwarding [Journal] // IEEE/ACM Transactions on Networking. - August 2005. - No. 4 : Vol. 13. - pp. 790-801.

SourceForge www.sourceforge.com [Online]. - 2010.

SPICE SPICE Overview. [Online]. - www.seas.upenn.edu/~jan/spice/spice.overview.html.

Stevens W.Richard UNIX network programming [Book]. - [s.l.] : Prentice Hall PTR, 1999.

Stroustrup B. The C++ Programming Language, Special Edition [Book]. - Reading, Massachusetts : Addison Wesley Longman, Inc, 2000.

Symbian Symbian website [Online]. - 2009. - http://www.symbian.com.

Tsui James Bao-yen Fundamentals of global positioning system receivers: a software approach [Book]. - [s.l.] : John Wiley & Sons, 2000.

Volz B. [et al.] DHC Load Balancing Algorithm [Report]. - [s.l.] : The Internet Society, 2001. - RFC 3074.

Wang J. [et al.] Intelligent Load Balancing Strategies for Complex Distributed Simulation Applications [Conference] // IEEE Computer Society, International Conference on Computational Intelligence and Security. - 2009. - pp. p.182-186.

Weighorst H., Hooper G. and Greenberg D Improved Computational Methods for Ray Tracing [Journal]. - [s.l.] : ACM Transactions on Graphics, 1984. - 1, January : Vol. 3.

www.artis-software.com Artis Software Corp., Torino, Italy [Online].

www.bbv-software.com BBV Software, Enchede, The Netherlands [Online].

www.virtualphotonics.com Virtual Photonics, Inc. Berlin, Germany [Online].

Zoltan Zoltan library, toolkit for parallel applications, homepage [Online]. - 2010. - 2010. - http://www.cs.sandia.gov/Zoltan.

VITA

Aleksandr Panchul was born in Kiev, USSR, where he completed Physics and Mathematics High School. In 1989 he moved to Moscow, Russian Federation. He received M.Sc. in Computer Science and Applied Mathematics from Moscow Institute of Physics and Technology in 1997, diploma with Honors. By that time he also participated in theoretical physics research at Uppsala University, Sweden. During his Masters program he worked at various software companies, writing operating system software and designing database systems. In period from 1997 to 2005 he was employed by top-level telecommunications and financial data market companies in New Jersey and New York City. In 2010 he worked with T1 line interfaces and ISDN protocol.

Aleksandr also is a member of Eta Kappa Nu (HKN) Association, Kappa Upsilon Chapter. In 2005 he started software project Computer Engineering Framework (CEF). In 2009 he was selected into Who's Who Among Students in American Universities and Colleges for outstanding accomplishments and excellence as a student at the University of Texas.

His research interests include, among others, software engineering, distributed systems, 3D animation, artificial intelligence, microcontrollers and robotics, digital communications, CPU emulators, and mobile applications.