ISSN 1727–6209

RESEARCH INSTITUTE FOR INTELLIGENT COMPUTER SYSTEMS
TERNOPIL NATIONAL ECONOMIC UNIVERSITY
and
V.M. GLUSHKOV INSTITUTE FOR CYBERNETICS,
NATIONAL ACADEMY OF SCIENCES, UKRAINE

International Journal of

Computing

Published since 2002

December 2013, Volume 12, Issue 4

Ternopil – 2013

International Journal of Computing

December 2013, Vol. 12, Issue 4

Research Institute for Intelligent Computer Systems, Ternopil National Economic University and V. M. Glushkov Institute for Cybernetics, National Academy of Sciences, Ukraine

Editorial Council

Address of the Journal

Research Institute for Intelligent Computer Systems
Ternopil National Economic University
3, Peremoga Square
Ternopil, 46020, Ukraine

Phone: +380 (352) 47-5050 ext. 12234
Fax: +380 (352) 47-5053 (24 hours)
computing@computingonline.net
computer.journal@gmail.com
www.computingonline.net

ISSN 1727–6209

НДІ ІНТЕЛЕКТУАЛЬНИХ КОМП'ЮТЕРНИХ СИСТЕМ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
і
ІНСТИТУТ КІБЕРНЕТИКИ ІМ. В.М. ГЛУШКОВА,
НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ

Міжнародний науковий журнал

Комп'ютинг

Видається з 2002 року

Грудень 2013, Том 12, Випуск 4

Постановою президії ВАК України № 1-05/3 від 14 квітня 2010 року міжнародний науковий журнал Комп'ютинг віднесено до переліку наукових фахових видань України, в яких можуть публікуватися результати дисертаційних робіт на здобуття наукових ступенів доктора і кандидата технічних наук

Тернопіль – 2013

Міжнародний науковий журнал Комп'ютинг

Грудень 2013, том 12, випуск 4

НДІ інтелектуальних комп'ютерних систем, Тернопільський національний економічний університет і Інститут кібернетики ім. В. М. Глушкова, Національна Академія наук України

Зареєстрований Міністерством юстиції України. Свідоцтво про державну реєстрацію друкованого засобу масової інформації – серія КВ №17050-5820ПР від 15.07.2010.

Друкується за постановою вченої ради ТНЕУ, протокол № 2 від 30 жовтня 2013 року

Редакційна рада

Адреса журналу

НДІ інтелектуальних комп'ютерних систем
Тернопільський національний економічний університет
площа Перемоги, 3
Тернопіль, 46020, Україна

Тел.: 0 (352) 47-50-50 внутр. 12234
Факс: 0 (352) 47-50-53
computing@computingonline.net
computer.journal@gmail.com
www.computingonline.net

CONTENTS

ЗМІСТ

EDITORIAL
"HIGH PERFORMANCE COMPUTING"

Guest Editors: Volodymyr Turchenko, Dora Blanco Heras

It's our pleasure to welcome you to read this thematic issue of IJC on High Performance Computing. Most of the papers (6 from 8) presented in this issue are the extended versions of the papers gathered from the Special Stream on High Performance Computing organized (first time) at the Seventh IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS'2013), which was held in Berlin, Germany, September 12-14, 2013.

The Conference was organized by the Research Institute for Intelligent Computer Systems, Ternopil National Economic University, Ternopil, Ukraine and co-organized by the University of Applied Sciences, Hochschule für Technik und Wirtschaft (HTW) Berlin, Germany.

The IDAACS Conference series is established once every two years as a forum for high quality papers on state-of-the-art theory, technology and applications of intelligent data acquisition and advanced computer systems. These techniques and applications have experienced a rapid expansion in recent years that has resulted in more intelligent, sensitive, and accurate methods of data acquisition and data processing. Subsequently, these advances have been applied to: manufacturing process control and inspection; environmental and medical monitoring and diagnostics; and intelligent information gathering and analysis for the purpose of security and safety.

The papers selected for this thematic issue reflect the variety of research in the area of high performance computing.

The paper "*A Parallel Template for Implementing Filters for Biological Correlation Networks*" by **Kathryn Dempsey, Vladimir Ufimstev, Sanjukta Bhowmick** and **Hesham Ali** deals with high throughput biological experiments in system biology used to analyze the state of cellular mechanisms on the broad scale. These experiments open possibilities for the scientific researcher to understand how multiple components come together, and what goes wrong in disease states. The problem is that the data returned from these experiments is massive and heterogeneous, and requires intuitive and clever computational algorithms for analysis. The authors have proposed the correlation network model as a tool for modeling and analysis of this high throughput data and the structures within the model to represent key players in major cellular pathways. The authors prove that network filtering using graph theoretic structural concepts can reduce noise and strengthen biological signals in these networks. However, the process of filtering biological networks using such filters is computationally intensive and the filtered networks remain large. The authors develop a parallel template for these network filters to improve runtime, and use high performance environment to show that parallelization does not affect the network structure or the biological function of that structure.

The paper "*Multiresolution rendering Based on GPGPU Computing*" by **Julián Lamas-Rodríguez, Francisco Argüello** and **Dora B. Heras** addresses the problem of visualizing large volumetric datasets while processing on the GPU. The authors state that the design of GPU's volume rendering solutions must deal with the limited available memory available in a graphic card. The authors present a system for multiresolution volume rendering which preprocesses the dataset dividing it into bricks and generating a compressed version by applying different levels of compression based on wavelets. The compressed volume is then stored in the GPU memory. For the later visualization process by texture mapping each brick of the volume is decompressed and rendered with a different resolution level depending on its distance to the camera. This approach computes most of the tasks in the GPU, thus minimizing the data transfers among CPU and GPU. The results obtained in the paper are competitive for volumes of size in the range between 64^3 and 256^3.

Oleksandr Sudakov, Andrii Salnikov, Ievgen Sliusar and **Oleksandr Boretskyi** within their paper "*Tools for Biomedical Data Archiving in Ukrainian Grid Infrastructure*" proposed, implemented and deployed tools for data archiving and extraction within Ukrainian National Grid for end-users' applications in medical imaging, non-linear dynamics and molecular biology. The proposed tools provide the facilities to utilize large distributed storage space in grid infrastructures for different practical tasks including desktop applications. The authors describe that (i) the tools may be successfully used even when it is impossible to setup grid middleware on client platforms, use web browser interfaces or grid security infrastructure authentication and (ii) the tools consist of extensible client compatible with different software and hardware platforms; web service for data transfer and web service for transparent data replication on grid storage elements.

The paper "*Big Data Transfer for Tablet-Class Machines*" by **Tevaganthan Veluppillai, Brandon Ortiz** and **Robert E. Hiromoto** presents a comparative

study of several well-known data transfer protocols to address the issue of big data transfer for tablet-class machines. The analyzed protocols include standard Java, C++ and block-data transfers protocols that use both the Java New Input Output and the Zerocopy libraries. The obtained experimental results are compared against the standard Java IO and C++ stream-based file transport protocols. The motivation for this study is the development of a client/server big data file transport protocol for tablet-class client machines that rely on the Java Remote Method Invocation package for distributed computing.

The paper "*Vector Clock Tracing and Model Based Partitioning for Distributed Embedded Systems*" by **Robert Hoettger, Burkhard Igel** and **Erik Kamsties** discusses tracking, partitioning and tracing in modern dynamic high performance computing systems with respect to distributed systems and proposes new mechanisms for an advanced utilization of software in this domain. The authors presented a specific tracking mechanism via vector clocks for model and code partitioning purposes and the determination of causality relations. Then the authors introduce a tracing approach for an effective analysis and utilization of code and the corresponding architecture. The combination of both approaches leads to a high degree of parallelism and a fine-grained structure of execution units, that further traced, supports a precise analysis of synchronous and asynchronous system's behavior as well as an optimal load balancing.

The paper "*Integration of Cloud Computing Platform to Grid Infrastructure*" by **Vladislav Falfushinsky, Olena Skarlat** and **Vadim Tulchinsky** proposes the integration approach that joints the principles of both grid and cloud computing. The described implementation of the cloud platform integration into grid-infrastructure has been tested and applied in operation within Ukrainian National Grid which is an integrated part of EGI. The paper describes the development of a set of commands which is sufficient for flexible command line interaction with the cloud platform integrated in grid. The main advantages of the proposed solution are: (i) quick deployment of new or alternative software versions within virtual organization, (ii) arbitrary mix of grid and cloud/grid tasks on the same clusters, (iii) dialog and on-line grid environments for immediate user's operations, (iv) automated data flow and distributed storing, (v) Linux/Windows portability and (vi) tolerance to differences in different operational environments.

The paper "*Hardware Models for Automated Partitioning and Mapping in Multi-Core Systems using Mathematical Algorithms*" by **Lukas Krawczyk** and **Erik Kamsties** introduces a hardware model which is capable to support automated partitioning and mapping in heterogeneous multi-core systems. The case study, Freescale multi-core CPU, showed how the developed hardware model is able to support the involved steps and which amount of hardware-related information is required for an automated execution. The paper outlines a feasible implementation of these aspects as a part of a seamless tool chain.

The paper "*Spot Price Prediction for Cloud Computing Using Neural Networks*" by **Volodymyr Turchenko, Vladyslav Shultz, Iryna Turchenko, Richard M. Wallace, Mehdi Sheikhalishahi, Jose Luis Vazquez-Poletti** and **Lucio Grandinetti** describes the commodity bidding approach for cloud computational resources on the case study of Amazon Elastic Cloud Computing (EC2) environment. Their analysis has shown that similar bidding methods exist for other cloud-computing vendors as well as multi–cloud and cluster computing brokers such as SpotCloud. The commodity bidding for computing resources has resulted in complex spot price models that have ad-hoc strategies to provide demand for excess capacity. The authors present a predictive model for future short-term and long-term spot price prediction using neural networks giving users a high confidence on future prices aiding bidding on commodity computing.

We hope the readers find the papers of this thematic issue interesting, useful and even enjoyable as well!

Dr. Volodymyr Turchenko
Associate Professor of Information Computing Systems and Control Department
Head of Neural Network and Parallel Computing Research Group
Research Institute for Intelligent Computer Systems
Ternopil National Economic University
3 Peremogy Square, 46020, Ternopil, Ukraine
e-mail: volodymyr.turchenko@gmail.com
http://uweb.deis.unical.it/turchenko/

Dr. Dora Blanco Heras
Associate Professor in the Department of Electronics and Computer Engineering
Centro Singular de Investigación en Tecnoloxías da Información
University of Santiago de Compostela
Rúa de Jenaro de la Fuente Domínguez
Santiago de Compostela, 15782, La Coruña, Spain
e-mail: dora.blanco@usc.es
http://citius.usc.es/equipo/persoal-adscrito/dora.blanco?language=en

computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

A PARALLEL TEMPLATE FOR IMPLEMENTING FILTERS FOR BIOLOGICAL CORRELATION NETWORKS

Kathryn Dempsey, Vladimir Ufimtsev, Sanjukta Bhowmick, Hesham Ali

College of Information Science and Technology, University of Nebraska at Omaha
E-mail: hali@mail.unomaha.edu

Abstract: High throughput biological experiments are critical for their role in systems biology – the ability to survey the state of cellular mechanisms on the broad scale opens possibilities for the scientific researcher to understand how multiple components come together, and what goes wrong in disease states. However, the data returned from these experiments is massive and heterogeneous, and requires intuitive and clever computational algorithms for analysis. The correlation network model has been proposed as a tool for modeling and analysis of this high throughput data; structures within the model identified by graph theory have been found to represent key players in major cellular pathways. Previous work has found that network filtering using graph theoretic structural concepts can reduce noise and strengthen biological signals in these networks. However, the process of filtering biological network using such filters is computationally intensive and the filtered networks remain large. In this research, we develop a parallel template for these network filters to improve runtime, and use this high performance environment to show that parallelization does not affect network structure or biological function of that structure.

Keywords: high performance computing, correlation networks, parallel computing, network filters, graph algorithms, noise, biological signal.

1. INTRODUCTION

High-throughput assays are now able to take surveys of the entire cellular landscape at once – be it gene expression, protein function, or any other experimentally quantifiable measure. The technological capacity for examining the minutiae on the grand scale is growing, and with it grows the need for analyses that are both computationally robust and informative. The inherent danger of these experiments and their post-completion analytics lies in the sea of information available. It is possible to find multiple needles in the proverbial haystack, and extremely difficult to discern which needle is the biological candidate for causing any observed phenotypical deviations from the norm, be that disease, aging, or some other biological phenomenon. Simply put, the increase in technological capacity is accompanied, then, by an increase in data heterogeneity, volume, and noise – leading to biological "big data" [10].

To accommodate these specific problem areas, the network model has been employed as an effective tool for data visualization and analysis. Among others, three of the major reasons why network modeling is becoming popular include (1) networks are easy to work with, (2) networks retain the ability to represent relationships between biological entities (not just the entities themselves), and (3) well-established graph theoretic approaches can be used on the network model for analysis. Graph theory has been around at least since the 1700's, ever since Leonhard Euler proposed his solution to the Seven Bridges of Königsberg Problem, and ever since, methods to iterate through and understand the graph model have been identified, solved, and analytically improved.

Consider, then, the problem posed by high-throughput experimentation: large sets of heterogeneous data contain multiple levels of information, (functional ontology, pathway information, gene to protein attributes, etc.), not all relevant to the research query at hand. The network model becomes an ideal tool for the analysis of these datasets, if used cleverly, and if the model is shown to provide useful information. Indeed, as Albert-László Barabási and his team first proposed in their sentinel 1999 work "Emergence of Scaling in Random Networks" and then again in their 2001 follow-up "Lethality and Centrality in Protein Networks," networks can be used to reveal important information about an organism on the

cellular level. In particular, the two publications mentioned established that the degree distribution of many real world networks, including the protein interaction network (PPI), followed a characteristic power-law distribution. In a protein interaction network, nodes are typically representative of proteins, and edges exist between two nodes if there is a measurable, physical interaction between two proteins. This power-law distribution proposed means that the network itself contains few nodes that are highly connected to others, and many nodes that are poorly connected. Since then, the network model in the biological realm has exploded in popularity, and other biological relationships to structural properties of the model have been found, for example, it is now established that within the PPI model, clusters of genes, particularly cliques or completely connected subnetworks, tend to represent protein complexes. The typical protein complex is a conglomeration of proteins that all interact together to perform some function, and will not function without "participation" of all its components. In this way, many new proteins required for cellular function have been identified.

Multiple network models have been proposed to represent biological data: the PPI, the metabolic network, the transcriptome, etc. While they all can be similar (and more importantly, aligned and integrated), there are inherent differences in each model type and benchmarking of the structures native to each model must be performed for that model to become useful to the research of systems biology [4]. In that regard, the correlation network, or a network where nodes represent genes and edges represent a correlation of expression pattern for those genes, is a type of network that is only recently becoming more well understood and finding popularity. Correlation networks have been found to mirror some of the major findings in biological network theory; for example, structures within these networks (hubs, clusters [8], etc.) can point to biological functions, and the relationships between those genes (which may previously may have been unknown). While these networks are increasing in popularity, the issue remains that networks are typically large and noisy [19], corrupting the biological signal behind observed phenotypes. As such, multiple methods for sorting signal from noise have been proposed. One such general method, network filtering, has found measurable success in reducing network size and noise while enhancing ability to identify relevant biological functions.

Previous work has shown that filters imposed on networks that represent gene co-expression (this co-expression can be coincidental or causative) are an effective means for removing "noisy" edges while enhancing biological signal. Duraisamy *et al.* [19] and Dempsey *et al*. [15-17] found that filters that augment networks such that edges that exist as part of a cycle (a path of connected nodes where the original node in the path and the terminal node in the path are the same) typically are found to represent noise. A filter, for example, can remove around 25 % of relationships from the original network, while also maintaining clusters that exist in the original network. The filter can also reveals clusters that were previously "hidden" or undiscoverable by common clustering algorithms due to density or neighborhood distortion. Dempsey *et al*. [17] explored how a maximum weighted spanning tree filter affects biological relevance of high degree or hub nodes in the correlation network. (Biologically relevant nodes in a correlation network can typically be expected to represent lethal nodes [18], or nodes that represent genes that when knocked out *in vivo* results in expiration of the organism at some early stage in development [14].) This study found that by using a spanning tree filter, it is possible to more accurately identify biologically relevant hub nodes in the correlation network due to the removal of coincidental edges. Further, a "hybrid" filter was created that incorporated a spanning tree and a chordal filter by adding edges back into the network. The focus of the study then became the examination of how the biological relevance of hub nodes is further enhanced (i.e., hub nodes from the original network gain more edges back, making them easier to identify as hub nodes). This filter incorporated edge re-addition in two steps, one where edges were added such that chordality is maintained, and a second where edges were added with a less strict condition – chordality is preferred, but not some larger cycles are allowed, if they are part of clusters. The best parameters from this study revealed that adding in edges that did not necessarily maintain chordality (but not adding in all edges) was best able to identify biologically relevant hub nodes. In short, we have three major versions of the network that we are able to test for biological relevance; these variations are shown in Fig. 1.

"Hub" nodes in correlation networks can be disassortative or assortative [27], the former indicating that its neighbors are poorly connected and the latter indicating that the hub is very well connected; in such cases the assortative hub can be found to exist within clusters as a member of a dense community. Results from Dempsey *et al.* [17] show that while the aforementioned maximum spanning tree (MAXST) filter is able to identify lethal hub nodes better than the original network (according to degree), the edge-addition methods are both better than the spanning tree only approach. We speculate that this is because the MAXST approach only identifies disassortative nodes within the network;

adding edges back in allows for the assortative hubs, which by definition require more edges between neighbors, makes identification of these hubs possible.

Theoretically speaking, a biological network is self-organizing and as contains multiple built-in redundancies to ensure survival in structural breakdown; this characteristic of self-organizing systems [1] is consistent with the need for clusters in a correlation network –it reflects the inherent need for a set of genes to be co-expressed and working in concert toward some discrete function.

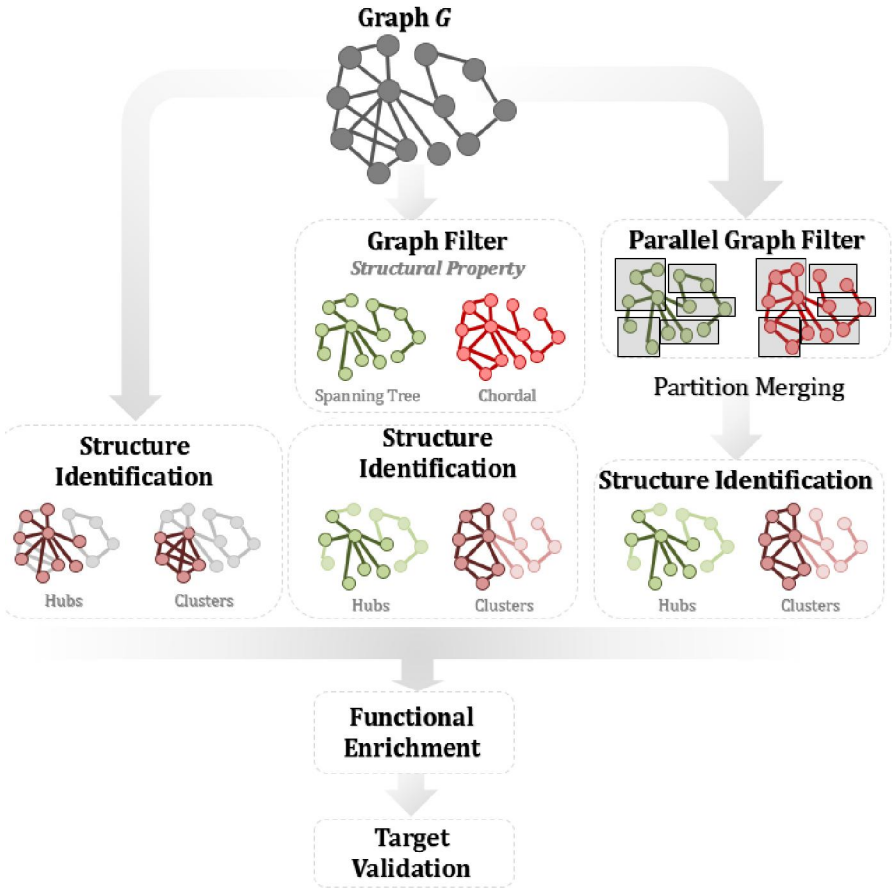


Fig. 1 – An overview flowchart of this approach.

In this study, we further examine the applicability of this hybrid filter by examining its effectiveness in enhancing clusters in correlation networks. Previous studies on chordal filters by [15-17] revealed that a chordal filter is able to maintain current clusters from the original network and identify new clusters that were previously hidden. Previous studies on the hybrid chordal filter have only examined its effectiveness in identifying biologically relevant hub nodes, not clusters. Therefore, in this study we implement and apply a parallel filtering approach to networks generated from an aging mouse gene expression study to show its effectiveness in identifying clusters and the speedup that results. An overview of our method is shown in Fig. 1. Previous work in this area of biological correlation network filters used relatively small networks, and as such it is crucial to show that these filters are able to scale and maintain the same result.

2. HYPOTHESIS

In this study, we further examine the applicability of the previously studied filters by examining their effectiveness in enhancing clusters and hubs in correlation networks. Gene expression correlation networks tend to get large when analyzed on the whole-genome scale, so it is important to be able to parallelize the process and also reduce runtime of the clustering method (typically the longest step in the analytic pipeline), while still being able to identify relevant biological clusters. From our previous results using network filters, we have observed the following phenomena:

- Chordal filters maintain network clusters [15]
- Spanning tree filters maintain lethal hub nodes [16]
- The hybrid filter maintains lethal hub nodes *best* when edges are added back into the

network without necessarily maintaining chordality [17]
- Almost all the chordal-like edges are added to the spanning tree in 1 to 2 iterations [8]

Based on these observations, we can propose our hypothesis for how well the hybrid filter is able to identify clusters, in addition to the rather straightforward hypothesis that the process of filtering networks should reduce their density and in turn reduce the search space and the computational time of extracting information from the networks.

- *H0a: There are significant independent tasks associated with graph-theoretic network filtering which implies that Parallelization of these filters results in decreased runtimes, indirectly leading to faster analysis.*
- *H0b: Filtered networks retain their relevant biological structures, including hubs and clusters.*
- *H0c: The changes due to parallelization do not significantly effect the utility of the filters.*

3. METHODS

3.1. NETWORK CREATION

Networks are created by using data from NCBI's Gene Expression Omnibus, which houses data from microarray, RNA-seq, and other high-throughput assays [2]. The data was taken from GSE8150 [5], which uses brain tissue from mice at ages 5 months and 30 months. Three datasets total resulted from the GSE8150 experiment:

1. Untreated mice at 5 months (YMBC)
2. Treated with α-tocopherol, 30 months (MOBATOP)
3. Treated with γ-tocopherol, 30 months (MOBAGTOP)

Briefly, mouse-aging networks were created using the pairwise correlation coefficient calculated for each pair of genes by measuring correlation of pattern expression. Genes or gene products are represented in the network as nodes. Correlation can fall between -1.00 and 1.00, and correlations passing hypothesis testing using the Student's T-test (p-val < 0.0005) are used to draw an edge between the representative nodes, with the weight of the edge set to the actual correlation score.

3.2. TEMPLATE FOR FILTERING ALGORITHMS

The primary goal of filtering algorithms used in our experiments is to preserve the structural properties of the networks that highlight the corresponding system properties. Specifically, we use maximum spanning tree (MAXST) filters to identify hubs, which represent lethal nodes and chordal graph based (CHD) filters to extract important communities in the networks. Both these filters, as well other structural sampling methods such as random walk, forest fire, breadth first search (BFS), etc. are based on network traversal. As part of our implementation we propose a template that can be easily modified for all such graph traversal-based sampling algorithms.

To create such a template, we observe that all graph traversal algorithms follow this pattern;

(i) Select a start vertex

(ii) Identify its neighboring nodes that have not been visited

(iii) Put a priority value to the neighboring nodes.

For example, the priority for BFS is the distance from the root; for MAXST using Prim's method [27] it is the neighbor with the highest weight; for CHD using Dearing's Algorithm [28] it is the neighbor with the most connections to the filter graph

(iv) Add the neighboring nodes to a maximum priority queue. Mark them as visited.

(v) Remove the top node from the priority queue

(vi) Add this node to the filtered network, if it maintains certain structural properties

For BFS and MAXST the network should remain acyclic. For CHD the size of a cycle cannot be more than three.

(vii) This new node becomes the start vertex.

(viii) The process is continued until all the vertices have been visited.

Biological networks can often have disconnected components (generally, one giant component and many small ones). This template for graph traversal can also be modified for disconnected components by adding a check to ensure that when a traversal ends, i.e. there are no more new vertices to add to the priority queue, there are also no unvisited vertices remaining. If an unvisited vertex is found, then it belongs to a new component, and it is selected as the next start node. Note that the only change required to implement a new traversal method is at steps (iii) and (vi). Thus this template facilitates easy modification and experimentation of new traversal techniques.

3.3. PARALLEL FILTERING ISSUES AND SOLUTIONS

We now discuss a parallel implementation for the traversal template. The template by itself does not lend easily to parallelization since the start nodes are selected sequentially from the priority queue one by one. Although BFS has a wave front like expansion of neighbors from which traversal can be done in parallel, this is a specialized case and does not hold

for CHD or MAXST. Another area of parallelization is when searching for the neighbors. However most biological networks are scale-free and therefore most of the nodes have low degree. Thus parallelizing the neighbors search will not significantly affect the running time.

We therefore decided to partition the network across different processing units (in this case, threads), execute the traversal for each partition separately and then combine the filters obtained from each partition. One issue in this method is that although the individual filters maintain the specified traversal properties, it is more difficult to ensure that the combined filter will also do so. In fact, attempting to maintain the BFS or MAXST tree or a chordal graph across the combined filters, often results in the combination process becoming sequential and extremely time consuming. As a compromise we decided to opt for quasi-filters instead of exact filters. This helps reduce the time and makes the combination process parallel as well. Moreover, the quasi-filters maintain most if not all of the properties of the exact filters and therefore do not affect the analysis results significantly.

The combination process for BFS and MAXST is as follows; add exactly one edge between partitions Pi and Pi+1, and no edge between the first and the last partition. It is easy to see that the resulting quasi-filter is still a tree. However, for BFS some of the nodes may not be in the shortest path from the root, as would be the case for an exact BFS tree and for MAXST the edges connecting two partitions may not be the ones with the maximum weight. In the combination process for CHD one node from partition Pi is connected to all its neighbors in Pi+1. If the neighbors are connected then the chordal graph property is maintained, otherwise we will end up with a few cycles of length greater than three. So long as the percentage of such larger cycle is significantly smaller than the number of triangles the benefits of the chordal graph are maintained. Note that since Pi connects to Pi+1 and Pi+1 connects to Pi+2, each of the combination steps can be done in parallel by each partition (except for the last partition which does not combine). This increases the scalability of the filter.

3.4. CLUSTERING

Clustering was performed using MGClus [5] under default parameters. MGClus aims to identifying clusters that exist in *large biological networks* and has been shown to perform well in PPI's, whose clusters tend to be dense and range from very small (5 nodes) to large. In particular, MGClus runs very quickly at the command line which is why it was chosen for these very large biological networks.

3.5. DESCRIPTION OF EXPERIMENTS

Experiments were performed in quadruplicate for each of the three networks (YMBC, MOBATOP, MOBAGTOP) to highlight the consistency of our approach. There are several parameters to be measured within this set of research. These major parameters are:

- *Filter*
  - *Node Selection*
  - *Filter Iteration*
- *Speedup*

The first measure that we will address is the filter itself – what type of filter is applied, and how many iterations of edges are added back in. Two sub-parameters of the filter itself are node selection – how nodes used to filter the network is selected. The node selection process for the initial tree can use a breadth-first-search (BFS) or maximum weighted spanning tree (MAXST), or a chordal filter (CHD). The chordal filter itself is not a node selection method per se, but it filters the network such that the final network model is a chordal subgraph of the original. The second sub-parameter of the filter is the augmentation of the network, which determines how edges are added back to the tree. Edges are added back only if they are present in the original network, by adding them between nodes at distance-2 in the tree. This operation creates triangles, which is required for chordal graphs. The constraints can be tight such that only chordal graphs are created or loose (quasi) where some larger cycles are allowed. This parameter also can be iterated over many times, or none. In this paper, we use iterations of 0-3, meaning that at iteration 0, no augmentation is performed, at 1, only one round of augmentation is performed, and at 2, two rounds are performed, and so on. In a recent paper by West *et* al., it was found that the filtered network rarely changes its inherent base structure after the first few iterations, so extension beyond 3 iterations should not be required. Finally, we compare each of these parameterizations sequentially and in parallel.

4. EXPERIMENTAL RESULTS

The goal of this study is to establish benchmarks for the time required to build and analyze networks, and establish how changes in parameters affects this runtime in sequential and parallel environments. The structure of our experiments and research in this manuscript follows as thus: the effect of filter on network size (3.1), the effect of filters on cluster count and overlap with original clusters (3.2), how scalable are the parallelizations (3.3.). Biological impact of these results is described using hubs (3.4.) and clusters (3.5).

4.1. FILTERED NETWORK SIZE

One of the first measures that can indicate the power of a filter is edge density. For each network, we measure the number of nodes present after filtering (n) and the number of edges present after filtering (e). If the network was completely connected, we know the total number of edges present would be equal to $E = n*(n-1)/2$, assuming no self-loops or multiple edges. Next, to determine edge density, we take the number of edges in the filtered network (e) divided by the total number of edges possible (E) and present it as a percentage to give the edge density. The edge densities of all networks during sequential runs are shown in Fig. 2.

Fig. 2 – Edge Density Results for Sequential Runs of each network for each filter. Edge density is represented on the y-axis, as Total Edges/ total Possible Edges *100. The x-axis represents the network, filter, and iteration used.

Typically, correlation networks tend to be sparse, and have low levels of edge density, but this does not necessarily mean the network is small or easily manageable. For example, the ORIG YMBC network (shown at bottom in green, Fig. 2), has an edge density of 0.11 %, meaning that the number of edges in the represents less than 1 % of the possible edges given the number of nodes. However, this low number is deceiving. The network has 43,021 nodes and 1,050,293 edges, which is too large for graphic visualization even with the most current network GUIs; a network of this size must be handled at the command line. This task might be challenging for a person not primarily trained as a bioinformatician or computer scientist.

It is clear from each of the three networks in Fig. 2 (MOBAGTOP at top in blue, MOBATOP at middle in red, and YMBC at bottom in green) that the MAXST filter, at every increasing iteration, reduces edge density drastically from the network.

This would be expected, as the filter is more stringent – originally it creates a tree which by nature has no cycles. The CHD filter, in which edges are added to increase the neighborhood connectivity where clusters exist, far better maintains original edge density with every increasing iteration, particularly at $i = 3$. The only iteration of CHD that is similar in edge density to the MAXST filter is for $i=0$; after this, we see a large jump for every filter and every network from $i=1$ and on. Whether or not this is beneficial will be revealed in the biological and clustering analysis.

4.2. CLUSTER COUNT AND OVERLAP

The clusters identified by MGClus are based on shared neighbors, and it should be noted that any clustering algorithm will perform differently based on its core techniques. MGClus was designed for large biological networks and was shown to perform well in random networks (does not identify noise as signal) and well in PPIs (identifies dense clusters based on shared neighborhoods and neighborhood topology). The results of the clusters in terms of cluster size are shown in Fig. 3. It is clear that in terms of clustering consistency, the CHD filter far outperforms MAXST; in fact, the MAXST filters behave in an unexpected fashion in that the number of clusters per network actually decreases with addition of edges. One might speculate that this occurs due to the fact that adding edges back in will create clusters where there were *dense* neighborhoods in the original network, creating actual neighborhoods instead of small groups of poorly connected nodes (resulting in the initial high number of clusters). Therefore, the smaller clusters in the earlier trees get merged.

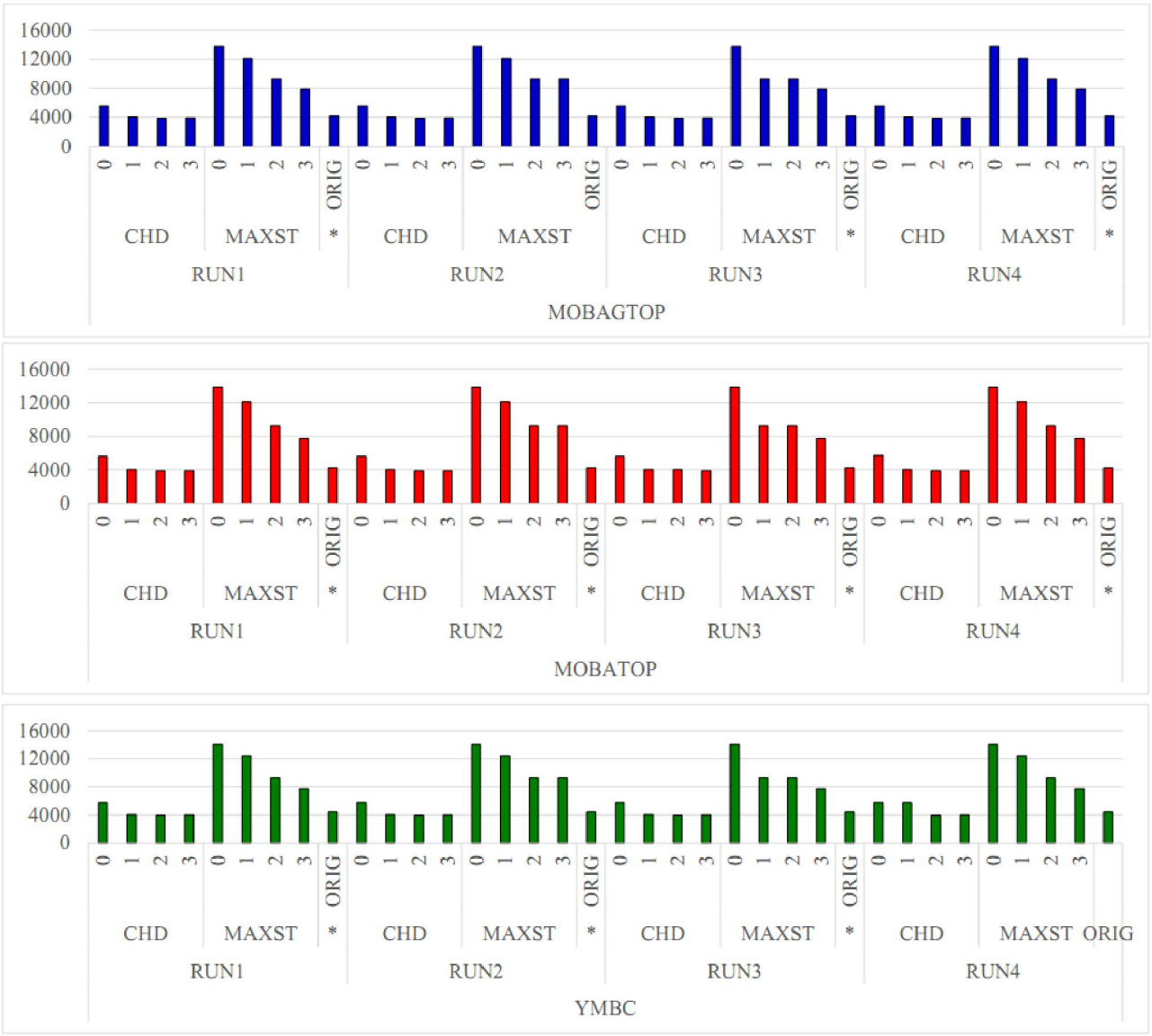


Fig. 3 – Cluster Count Results for Sequential Runs of each network for each filter. Clusters found per network is represented on the y-axis. The x-axis represents the network, filter, and iteration used.

One of the methods used to compare how well clusters in original networks are also identified in filtered networks is through cluster overlap. To determine cluster overlap, a list of each cluster in the original network and a list of each cluster in the filtered network was compared. The comparison or score (*Cluster Overlap Score)* was determined by checking the node overlap of clusters. For each cluster in original network G_O, each node was placed in a hash H_O. Then for each cluster in the filtered network G_F, each node was placed in a hash H_F. The two hashes are then compared, and if a node occurs in both clusters, it is scored as a match. The final match score for each cluster comparison is defined as the number of matches divided by the *total number of nodes in the original cluster*. Each original cluster is compared to each filtered cluster, and the *Maxscore* is defined as the highest final match score for that original cluster, or the overlap score of the original cluster compared and the highest overlapping cluster in the filtered network according to their node comparison. Note that the higher the *Maxscore*the better. *Maxscores* with a value greater than 1 indicate that multiple clusters within the filtered network overlapped with the original cluster, and overlaps are counted if the *Maxscore* of each of those clusters is equal. A *Maxscore* with a value greater than 1 is considered a sign of robustness of the cluster and the filter.

The *Maxscore*s of each network at RUN1 are shown in ranked order (lowest to highest) in Fig. 3. (Runs 2-4 are not shown as the filtered networks are extremely similar to RUN1). In Fig. 3, it is evident that there is a wide range of *Maxscores* for each original cluster, but it is quite evident that the CHD filters (at each iteration 0,1,2, and 3) are better performers than the MAXST filters in terms of finding clusters that are robust and having good overlap.

4.3. PARALLEL RESULTS AND SCALABILITY

For each of the two filters (CHDand MAXST) the scalability results are shown in Fig. 4. The machine used was a 64-bit with two physical Quad-Core AMD Opteron Processors (2394.112 Mhz CPU) with 32 GB of RAM per processor. The number of threads used ranges from 1 to 16 (specifically, the data points are for 1,2,4,8, and 16 threads) and the run time is just for the parallel portion of the algorithm not for the I/O of reading in the file and writing out the results.

After 16 threads, in each network, the partition becomes too small for effective parallelization and the overhead from the communication and combining all of the parts back together dominates the runtime.

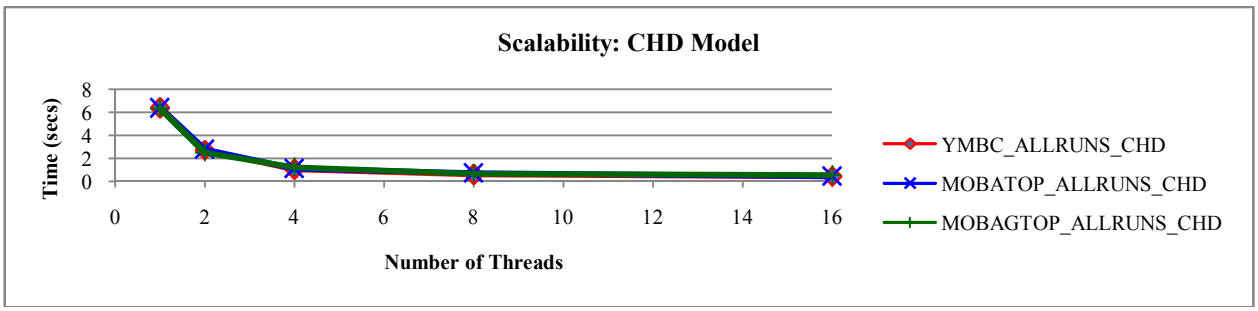


Fig. 4 – Scalability Results for the three networks with no iterations per CHD filter.

For each model there are 3 charts (MOBATOP, MOBAGTOP, and YMBC) and each chart shows the runtimes for different numbers of threads for each version of the network. For example, for the CHD model the chart for MOBATOP contains runtimes for MOBATOP_RUN1, MOBATOP_RUN2, MOBATOP_RUN3, and MOBATOP_RUN4. Table 1gives the number of vertices and edges in each network used.

Table 1. Sizes of networks tested.

Network	Vertices	Edges
MOBATOP	44577	2026962
MOBAGTOP	44564	1987326
YMBC	44875	2100586

Chordal Model. As is seen in Fig. 4, the CHD model performs well on all variants of each network. Notice that the shape of each of the curves is close to perfect scaling. When graphed using a log-log plot,

the curve actually becomes a straight line (with slope close to 1) meaning that when the number of threads is doubled, the runtime is cut in half. In fact, when scaling from 1 thread to 2 threads, the runtime is reduced by more than one half in every case. The same is true when comparing the runtimes for 2 and 4 threads. However, for 8 and 16 threads the runtimes do not get reduced as much and after 16 threads they begin to increase indicating that there is no benefit to increasing the number of threads after 16. *MAXSTModel*. The MAXST model (as shown in Fig. 5) performs the best out of the two models on all variants of each network. Notice that the shape of each of the curves is closer to perfect scaling than for the CHD model. When graphed using a log-log plot, the curve becomes a straight line (with slope greater than 1) meaning that when the number of threads is doubled, the runtime is cut by more than half. When doubling the number of threads starting from 1 all the way up to 16 threads the factor by which the runtime is reduced is more or less the same (around 60 %) i.e. runtime is cut down by the same proportion each time the number of threads are doubled (up to 16).

However, as was seen in the CHD model, after 16 threads the runtimes increase indicating that for networks of this size, it makes no sense to increase the number of threads past 16.

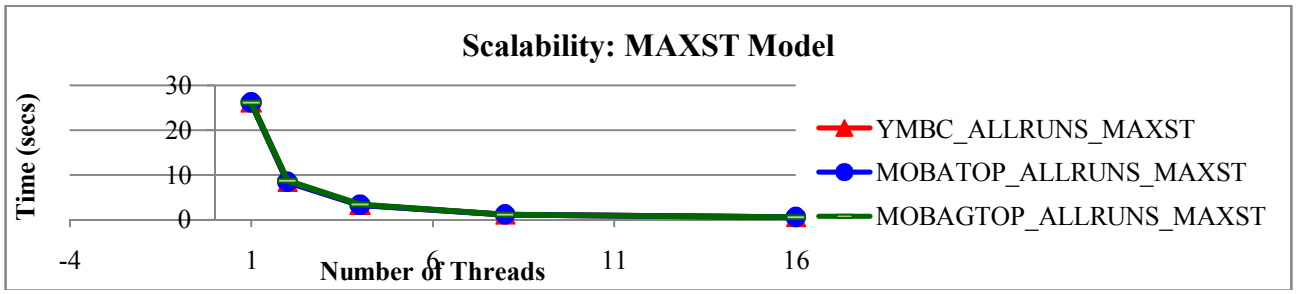


Fig. 5 – Scalability Results for the three networks with no iterations per MAXST filter.

4.4. BIOLOGICAL RESULTS – CLUSTERS

Due to the large amount of networks and subsequently, clusters generated, (over 777,000), biological significance of each will not be performed. Previous studies have shown that the chordal filters tend to maintain and/or enhance the biological function of found clusters if such a function exists. To highlight this, one example of cluster overlap is taken from the parallel code: cluster 2 from RUN4 of the YMBC original network was found to have a 61 % overlap with cluster 4037 from RUN4 of the YMBC CHD-1 filtered network. Between the two clusters, 20 nodes were found to overlap and 39 were unique to either the original or filtered network. A list of nodes found in both clusters is shown in Table 2.

The Gene Ontology profiles for both sampled clusters are shown in Fig. 6. .

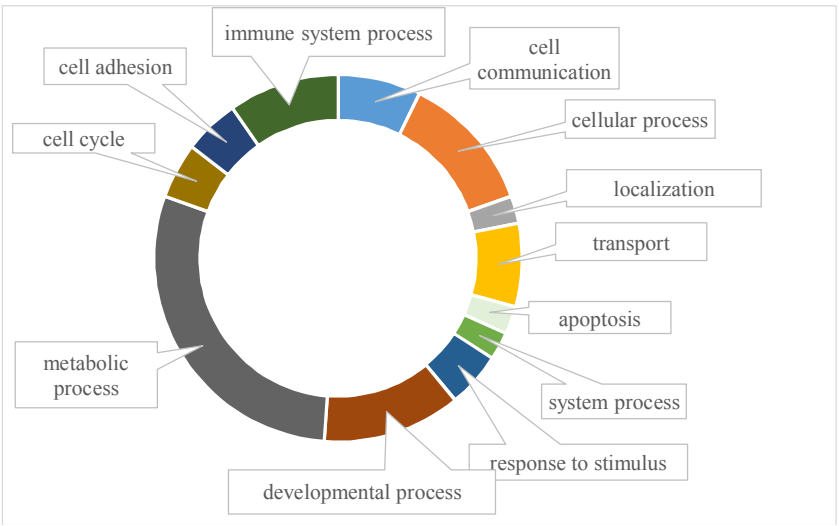


Fig. 6 (a). Gene Ontology profiles (Biological Process tree) for the sampled clusters from the original network.

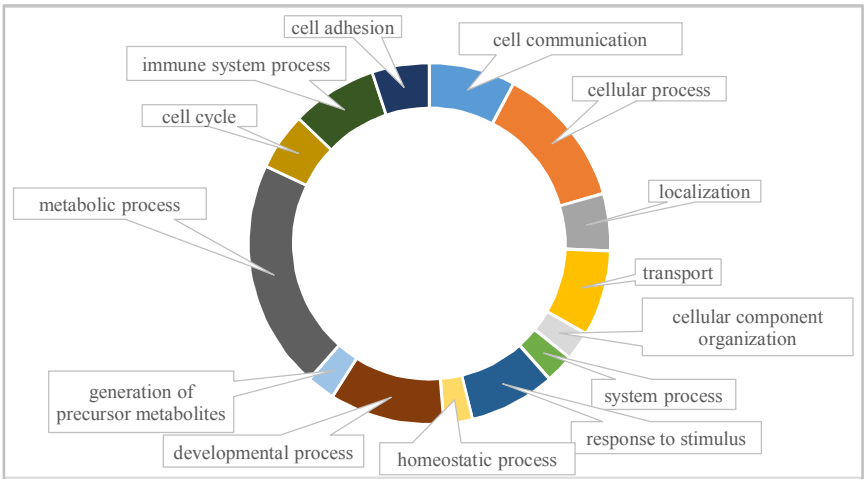


Fig. 6 (b). Gene Ontology profiles (Biological Process tree) for the sampled clusters from the filtered network.

Table 2. Node IDs, MGI IDs, and Gene Symbols for Original and Filter sample clusters from the YMBC RUN4 networks.

ORIGINAL CLUSTER			**FILTERED CLUSTER**		
Input	*ID*	*Symbol*	*Input*	*ID*	*Symbol*
1443866_at	MGI:2442106	Lrtm1	1429534_a_at	MGI:1923864	Immt
1451208_at	MGI:2385071	Etfl	1443866_at	MGI:2442106	Lrtm1
1460331_at	MGI:1915309	Tm9sf2	1453367_a_at	MGI:1923442	Abhd12
1422621_at	MGI:894323	Ranbp2	1422621_at	MGI:894323	Ranbp2
1453367_a_at	MGI:1923442	Abhd12	1457846_at	MGI:1917052	Cox11
1438511_a_at	MGI:1913464	Rgcc	1416514_a_at	MGI:1352745	Fscn1
1456035_at	MGI:2685230	Nxf3	1451655_at	MGI:2672859	Slfn8
1435569_at	MGI:2143561	D630029K05Rik	1424360_at	MGI:2384576	Tti2
1427672_a_at	MGI:1095419	Kdm6a	1435569_at	MGI:2143561	D630029K05Rik
1429534_a_at	MGI:1923864	Immt	1421233_at	MGI:1201409	Pknox1
1417086_at	MGI:109520	Pafah1b1	1430307_a_at	MGI:97043	Me1
1434508_at	MGI:1917343	Ube2q1	1460331_at	MGI:1915309	Tm9sf2
1437086_at	MGI:96919	Ascl1	1451208_at	MGI:2385071	Etfl
1421815_at	MGI:2145369	Epdr1	1417086_at	MGI:109520	Pafah1b1
1453214_at	MGI:1921738	Lrrc15	1450080_at	MGI:1920115	Cxx1c
1421874_a_at	MGI:1928138	Mrps23	1460726_at	MGI:87948	Adss
1451655_at	MGI:2672859	Slfn8	1419703_at	MGI:1858212	Col5a3
1457846_at	MGI:1917052	Cox11	1422409_at	MGI:104877	Hes3
1416514_a_at	MGI:1352745	Fscn1	1421878_at	MGI:1346862	Mapk9
1424360_at	MGI:2384576	Tti2			
1420598_x_at	MGI:99592	Defa-rs2			
1418751_at	MGI:1889342	Sit1			
1421233_at	MGI:1201409	Pknox1			
1422409_at	MGI:104877	Hes3			
1422699_at	MGI:87998	Alox12			
1421878_at	MGI:1346862	Mapk9			

While there are a few differences in the cluster profiles (apoptosis present in the Original cluster only, and three terms – generation of precursor metabolites, homeostatic process, and cellular component organization – are present in the Filtered cluster only) – the two clusters largely have a similar ontological profile, even considering that the filtered cluster has 7 less genes than the original and out of both clusters, only 20 genes overlap. (The Original cluster has 26 genes and the Filtered has 19). This highlights how we are able to maintain the biological integrity of the original network structure while reducing network size, cluster size, and noise. A level of discovery is even added with the finding of three new possible functions performed by the noted gene cluster.

An example of a cluster with low overlap to an original cluster can be found in the YMBC network using the CHD-1 filter. The first cluster found in this network contains 42 genes but has just 30 % overlap with any original clusters, meaning that this cluster is one that has been "found" or revealed with the removal of noise. This cluster was analyzed for functional enrichment using the GeneTrail [26] tool using default parameters. The cluster was found enriched in three main biological processes (P-val<0.05): cell development, cellular component morphogenesis, and cell morphogenesis. The Gene Ontology subtree for these functions is presented in Fig. 7 below. This is just one example of how a cluster that is not found in the original network can be discovered in the filtered network, or how noise removal can strengthen biological signal.

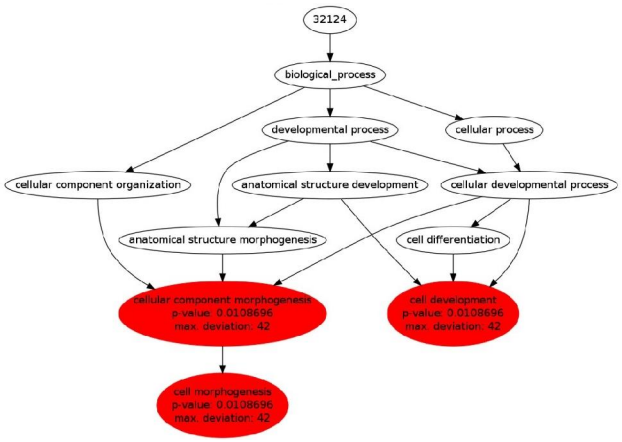


Fig. 7 – TheGene Ontology tree generated by GeneTrail Express [28] after enrichment of the second cluster of the filtered CHD-1 YMBC network.

5. DISCUSSION

In this study, we have examined how well our hybrid filter identifies dense clusters with high-degree nodes and biologically relevant nodes in correlation networks. It has been shown previously that network filters can remove noise from biological networks. The filter presented here can identify chordal or maximum spanning tree subgraphs, and also has the ability to add edges from the original network back in to bias that network toward dense communities. The speedup curves of the parallelized filter highlights the necessity of this parallelization, proving our original hypotheses, *H0a,* that *there are significant independent tasks associated with graph-theoretic network filtering which implies that Parallelization of these filters results in decreased runtimes, indirectly leading to faster analysis.* Secondly, the networks were shown to be consistent in terms of filter edge removal and cluster identification across multiple networks and multiple runs, proving our third hypothesis, *H0c*: That t*he changes due to parallelization do not significantly effect the utility of the filters.* Finally, we use examples of biological relevance to highlight our second hypothesis, showing that biological structures and their meanings are not affected by the parallelization of the filters. As the amount of information that needs to be incorporated into the network model grows, this parallel template can be trusted to improve computational runtimes for a faster and robust analysis.

6. ACKNOWLEDGMENT

This publication was made possible by the College of Information Science and Technology, University of Nebraska at Omaha and Grant P20 RR16469 from the NCRR, a component of the National Institutes of Health.

7. REFERENCES

[1] A. L. Barabasi, & Z. N. Oltvai. Network biology: Understanding the cell's functional organization, *Nature Reviews Genetics*, (5) 2 (2004), pp. 101-113.

[2] T. Barrett, S. E. Wilhite, P. Ledoux, et al. NCBI GEO: archive for functional genomics data sets – update 2013, *Nucleic Acids Research*, (41(D1), (2013), pp. D991-D995.

[3] O. Frings, A. Alexeyenko, and E. L. Sonnhammer, MGclus: network clustering employing shared neighbors, Molecular Biosystyems, (9) 7 (July 2013), pp. 1670-1675.

[4] J. Reichardt, Structure in Complex Networks, *Lecture Notes in Physics*, Vol. 766, Springer, Berlin, 2009.

[5] E. Reiter, Q. Jiang, and S. Christen, Anti-inflammatory properties of alpha- and gamma-tocopherol, *Molecular Aspects in Medicine*, (28) 5-6 (October-December 2007), pp. 668-691.

[6] What is Big Data | Big Data Explained, Villanova University, n.p. n.d., January 5, 2014 www.villanovau.com/university-online-programs/what-is-big-data/

[7] S. West, K. Dempsey, S. Bhowmick, and H. Ali. Analysis of incrementally generated clusters in biological networks using graph-theoretic filters and ontology enrichment, *IEEE International Conference on Data Mining (ICDM 2014)*, Dallas, Texas, USA, December 7-10, 2013.

[8] G. D. Bader, and C. H. W. Hogue. An automated method for finding molecular complexes in large protein interaction

networks, *BMC Bioinformatics*, (4) 2 (January 2003), pp. 2.

[9] C. J. Bult, J. T. Eppig, J. A. Kadin, J. E. Richardson, J. A. Blake et al. The Mouse Genome Database (MGD): mouse biology and model systems, *Nucleic Acids Ressearch*, (36) Database Issue (January 2003), pp. D724-D728.

[10] K. Dempsey, K. Duraisamy, H. Ali, & S. Bhowmick. A parallel graph sampling algorithm for analyzing gene correlation networks, *Proceedings of the 2011 International Conference on Computational Science*, Vol. 4, 2011, pp. 136-145.

[11] K. Dempsey, K. Duraisamy, S. Bhowmick, and H. Ali. The development of parallel adaptive sampling algorithms for analyzing biological networks, *Proceedings of the 11th IEEE International Workshop on High Performance Computational Biology (HiCOMB 2012)*, May 21, 2012, www.hicomb.org/proceedings.html

[12] K. Dempsey, S. Bhowmick, and H. Ali. Function-preserving filters for sampling in biological networks, *Proceedings of the 2012 International Conference on Computational Science*, Vol. (9), 2012, pp. 587-595.

[13] K. Dempsey, and H. Ali. On the discovery of cellular subsystems in correlation networks using centrality measures, *Current Bioinformatics*, (7) 4 (2012), publication pending.

[14] K. Duraisamy, K. Dempsey, H. Ali, and S. Bhowmick. A noise reducing sampling approach for uncovering critical properties in large scale biological networks, *Proceedings of the High Performance Computing and Simulation International Conference (HPCS)*, July 4-8, 2011, pp. 721-728.

[15] J. Dong, & S. Horvath, Understanding network concepts in modules, *BMC Systems Biology*, (1) 24 (June 1, 2007).

[16] W. J. Ewens, & G. R. Grant, *Statistical Methods in Bioinformatics*, 2nd edition, New York, NY: Springer, 2005.

[17] R. Edgar, M. Domrachev, and A. E. Lash. Gene expression omnibus: NCBI gene expression and hybridization array data repository, *Nucleic Acids Research*, (30) 1 (2002), pp. 207-210.

[18] A. J. Enright, S. Van Dongen, C. A. Ouzounis. An efficient algorithm for large-scale detection of protein families, *Nucleic Acids Research*, (30) 7 (2002), pp. 1575-1584.

[19] H. Jeong, S. P. Mason, A. L. Barabasi, & Z. N. Oltvai. Lethality and centrality in protein networks, *Nature*, (411) 6833 (2001), pp. 41-42.

[20] R. Linkser, Self-organization in a perceptual network, *Computer*, (21) 3 (1998), pp. 105-117.

[21] M. E. J. Newman. Assortative mixing in networks, *Physical Review Letters*, (89) 20 (October 2002), pp. 208701.

[22] R. Opgen-Rhein, & K. Strimmer. From correlation to causation networks: A simple approximate learning algorithm and its application to high-dimensional plant gene expression data, *BMC Systems Biology*, (1) 37 (August 2007).

[23] M. Verbitsky, A. L. Yonan, G. Malleret, E. R. Kandel, T. C. Gilliam, & P. Pavlidis. Altered hippocampal transcript profile accompanies an age-related spatial memory deficit in mice, *Learning & Memory*, (11) 3 (2004), pp. 253-260.

[24] A. Subramanian, P. Tamayo, V. K. Mootha, S. Mukherjee, B. L. Ebert, M. A. Gilette, A. Paulovich, S. L. Pomeroy, T. R. Golub, E. S. Lander, and J. P. Mesirov. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wise expression profiles, Proceeding of the National Academy of Sciences, (102) 43 (2005), pp. 15545-15550.

[25] D. Gleich. Gaimc: Graph Algorithms in Matlab Code. 16 May 2009. Mathworks.com. Obtained on 01.17.2012, from http://www.mathworks.com/matlabcentral/fileexchange/24134.

[26] A. Keller, C. Backes, M. Al-Adwahi, A. Gerasch, J. Keuntzerm, O. Kohlbacher, M. Kaufmann, and H. P. Lenhof. GeneTrailExpress: a web-based pipeline for the statistical evaluation of microarray experiments, *BMC Bioinformatics*, (9) 552 (December 2008).

[27] R. C. Prim. Shortest connection networks and some generalizations, *Bell System Technical Journal*, (36) (1957), pp. 1389–1401.

[28] P. M. Dearing, D. R. Shier, and D. D. Warner, Maximal chordal subgraphs, *Discrete Applied Mathematics*, (20) 3 (1988), pp. 181-190.

***Dr. Kate Dempsey**: Research Associate in the School of Interdisciplinary Informatics within the College of IS&T at UNO. She has been a member of the UNO Bioinformatics Research Group since September 2005, and has been involved with the UNO Bioinformatics Core Facility since its inception. She has publications in peer-reviewed conferences and journals in the areas of motif detection, parallel*

computing, and biological networks in systems biology. Her doctoral research involved using network theory and high-throughput biomedical data to identity structure-function relationships in a variety of graph theoretic data models. As a research associate, she has sought to expand her research from systems biology to complex systems, applying her knowledge of the biological world to the social world. In this time, she has collaborated with top companies in the Omaha metro-area to manage, analyze, and predict anomalies within masses of data, regardless of system type.

***Vladimir Ufimtsev** is a Ph.D. in IT student in the College of Information Science & Technology at the University of Nebraska at Omaha. He received his M.S. degree in Mathematics from Northeastern University (2009) and B.Sc. degree in Mathematics and Physics from University of Nebraska at Omaha (2006). His current research focus is on complex network analysis, high performance computing, efficiently identifying important vertices/edges in a network, group testing, the effects of network noise, and combinatorial algorithms. He has worked with the UNO Bioinformatics Research Group since 2011.*

***Sanjukta Bhowmick** is an Assistant Professor in the College of Information Science and Technology at the University of Nebraska at Omaha. She received her Ph.D. from the Pennsylvania State University. Her core research area is in high performance computing with a focus on the synergy of combinatorial and numerical methods. Her current projects focus on designing parallel, efficient and robust algorithms for analyzing large-scale dynamic networks. In particular, she is interested in evaluating how factors such as the discrepancy between the network model and the application and permutations in the input affect network analysis results and developing algorithms to minimize the influence of this noise.*

***Hesham H. Ali** is a Professor of Computer Science and the Lee and Wilma Seaman Distinguished Dean of the College of Information Science and Technology (IS&T), at the University of Nebraska at Omaha (UNO). He currently serves as the director of the UNO Bioinformatics Core Facility that supports a large number of biomedical research projects in Nebraska. He has published numerous articles in various IT areas including scheduling, distributed systems, data analytics, wireless networks, and Bioinformatics. He has also published two books in scheduling and graph algorithms, and several book chapters in Bioinformatics. He is currently serving as the PI or Co-PI of several projects funded by NSF, NIH and Nebraska Research Initiative (NRI) in the areas of data analytics, wireless networks and Bioinformatics. He has been leading a Research Group at UNO that focuses on developing innovative computational approaches to classify biological organisms and analyze big bioinformatics data. The research group is currently developing several next generation data analysis tools for mining various types of large-scale biological data. This includes the development of new graph theoretic models for assembling short reads obtained from high throughput instruments, as well as employing a novel correlation networks approach for analyzing large heterogeneous biological data associated with various biomedical research areas, particularly projects associated with aging and infectious diseases. He has also been leading two funded projects for developing secure and energy-aware wireless infrastructure to address tracking and monitoring problems in medical environments, particularly to study mobility profiling for healthcare research.*

computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

MULTIRESOLUTION RENDERING BASED ON GPGPU COMPUTING

Julián Lamas-Rodríguez, Francisco Argüello, and Dora B. Heras

CITIUS, University of Santiago de Compostela, Spain
{julian.lamas, francisco.arguello, dora.blanco}@usc.es

Abstract: The problem of visualizing large volumetric datasets is appealing for computation on the GPU. Nevertheless, the design of GPU volume rendering solutions must deal with the limited available memory in a graphics card. In this work, we present a system for multiresolution volume rendering which preprocesses the dataset dividing it into bricks and generating a compressed version by applying different levels of compression based on wavelets. The compressed volume is then stored in the GPU memory. For the later visualization process by texture mapping each brick of the volume is decompressed and rendered with a different resolution level depending on its distance to the camera. This approach computes most of the tasks in the GPU, thus minimizing the data transfers among CPU and GPU. We obtain competitive results for volumes of size in the range between 64^3 and 256^3.

Keywords: compressed volume rendering, texture mapping, multiresolution rendering, wavelet transform, quantization, CUDA, OpenGL.

1. INTRODUCTION

The evolution from graphics-specific accelerators to programmable vector processors has made of GPUs a standard platform for rendering volumetric datasets. However, recent years have witnessed significant improvements in the data acquisition methods, and, as a result, the size of datasets has increased. This poses a challenge given the limited memory resources available in current graphics hardware, and although each new GPU generation expands its memory capacity, the current trend shows that this problem will continue to exist in the future [1]. In this context, compression stands as an effective solution for processing increasingly larger datasets in the GPU. The compression is usually computed on a previously decomposed version of the volume.

As it is usual in the context of volume rendering, the volume is initially decomposed into a set of non-overlapping blocks, usually called bricks, so a single brick fits into the memory of the GPU. Bricks are compressed with multiple levels of compression. When the visualization process begins the bricks are loaded and rendered one at a time. In our implementation we use a single OpenGL 3D texture buffer to store the contents of a brick of data, and this buffer is reused every time a new brick is processed. The final visualization is achieved through texture mapping (also known as texture slicing) [9], which, along with ray casting, is one of the most popular methods to render volume data.

In this rendering technique, the 3D texture is mapped onto a proxy geometry composed of planar polygons that constitute camera-oriented translucent slices, i.e., the volumetric object is cut into slices that are rendered always parallel to the image plane [10, 11].

Several techniques of compressed volume rendering that can be found in the literature rely on storing the compressed volume data in a memory space different from the GPU (as the CPU main memory or a hard disk) [12, 13]. These out of-core techniques require transfer decompressed portions of the volume to the GPU memory before they can be rendered. Their performance is limited, at least, by the transfer rate of the PCI bus (e.g., 8 GB/s for PCIe 2.0).

A wide variety of approaches have been developed to build a compact representation of the data. In volume rendering, these solutions are usually asymmetric, i.e., the original dataset is decomposed and compressed in an off-line process which is executed only once without execution time constraints, while the decompression and visualization processes are executed in real time. Common methods for data compression may involve applying wavelet transforms [2, 3], vector quantization [4-6], or a multiscale tensor approximation [7]. For a survey on compressed

GPU-based volume rendering, we refer the reader to [8].

In this work, we have used a wavelet transform to compress the volume into a compact hierarchical form. We have selected the *Haar* wavelet, as it is computationally simple and very effective for fast reconstruction. Most of the coefficients of this transform are computed as sample-to-sample differences of the original volume data. This means that these coefficients will be of a small magnitude, or even zero, and therefore can be neglected without any significant loss of information. Our encoding scheme, a generalization of [2], benefits from this characteristic to obtain a more compact format of the compressed volume.

In this paper, we present a solution that stores the volume in the GPU memory in its compressed form. We couple decompression and rendering by dividing the volume in bricks which are processed one at a time, benefiting from the higher transfer rate of the GPU memory bus (192.4 GB/s in an NVIDIA GTX 580). Additionally, as our encoding scheme uses a wavelet transform, it supports decompressing bricks at different levels of resolution. The final rendering is executed using the texture mapping technique. We have obtained high speedups for the CUDA implementation of the steps of the algorithm. The complete system performs at an interactive and stable frame rate independent of the viewport size, while keeping a good compression ratio with a high visualization quality. This work is an extension of [9] where the rendering system was presented.

The rest of this paper is organized as follows. Section 2 describes the GPU architecture. Section 3 examines the design of our GPU-based system for compressed volume rendering. Section 4 analyzes the experimental results and compares our implementation to other similar works. Finally, Section 5 concludes discussing the main contributions and future work.

2. GPU ARCHITECTURE

GPUs are programmable architectures consisting of several many-core processors capable of running hundreds of thousands of threads concurrently. In this section we present a brief overview on the Fermi GPU architecture [15], which we have used to test our implementation of a GPU volume-rendering system.

NVIDIA's CUDA architecture [16] consists of a huge number of cores (or streaming processors, SPs), grouped into a set of streaming multiprocessors (SMs), with a very high memory bandwidth. As an example of this architecture, the GeForce GTX 580 has 16 SMs with 32 SPs each, resulting in 512 cores.

The programming model encourages a fine-grained level of parallelism within the single program multiple data (SPMD) paradigm [17]. A CUDA program (called *kernel*) is run by a *grid of threads*, which are grouped in *thread blocks*. Programmers can configure the size and distribution of the grid to their convenience and according to the requirements of the tasks to compute.

The architecture features several memory spaces. The *global memory* and *texture memory* spaces are accessible by the GPU, and also by the CPU through the PCI bus. Other memory spaces are located inside the chip, and provide a much lower latency: a read-only *constant memory*, a *shared memory* (which is private for each SM), a *texture cache* and, finally, a two-level cache that is used to speed up accesses to the global memory.

Coordination between threads within a kernel is achieved through *synchronization barriers*. However, as thread blocks run independently from all others, their scope is limited to the threads within the thread block.

3. THE RENDERING SYSTEM

Our solution involves two different stages: compression and visualization. The compression is executed on the CPU to preprocess the data generating the compressed volume from the original dataset.

The visualization stage runs on the GPU, and shows on the screen the reconstructed volume with different resolution levels depending on the distance to the camera. The visualization stage consists of two steps: a reconstruction of the volume followed by the rendering itself. This process is performed brick by brick with the required level of decompression for each brick.

Fig. 1 shows the different data structures used in this implementation. The original volume data is divided into bricks and each brick is divided into blocks of 16 × 16 × 16 elements. In the example shown in the figure a brick contains 2 × 2 × 2 blocks. Each block is divided into cells, each cell containing 4 × 4 × 4 elements. Finally, a chunk contains a group of 2 × 2 × 2 elements.

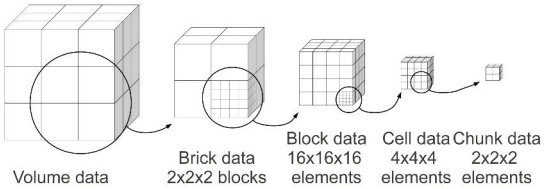


Fig. 1 – Data structures used in the rendering system.

Our compression algorithm requires reorganizing the volume data into smaller structures called

blocks, cells and chunks. For example, the wavelet transform that is applied to blocks processes data in a chunk basis. During the visualization stage, the volume is considered to be divided in bricks, which are processed individually.

The different steps of the initial compression and the later visualization stages will be described in the next subsections.

3.1. Compression

The compression stage is the preprocessing that takes place before the visualization process. Fig. 2 shows the different steps performed during this stage, and the data generated at each step. First, a wavelet transform is applied to the volumetric data. Afterwards, the wavelet coefficients are quantized to restrict the values to a limited number of possibilities. The quantization step scales down the coefficients obtained by the wavelet transform, nullifying those with a close-to-zero value, so losing information. Finally, the encoding step generates the compressed volume data, which is stored later in the hard disk. All these steps are executed on the CPU.

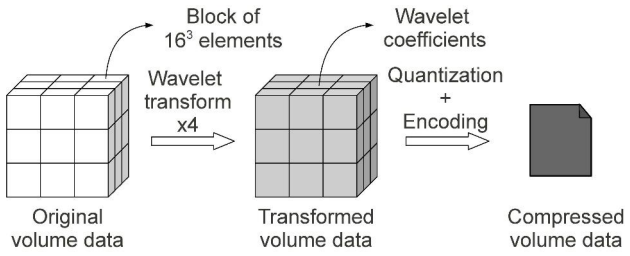


Fig. 2 – Compression steps over the original volume.

The wavelet transform step applies a wavelet-transform operator to blocks of 16 × 16 × 16 elements using a Haar filter. Our CPU implementation is similar to other solutions that can be found in the literature [2]. The transform is recursively applied to each block, generating bands of coefficients.

Fig. 3 shows how the wavelet transform generates the coefficients for a 16 × 16 × 16 block, which are then grouped in eight bands. These bands are labeled from LLL to HHH.

The LLL band contains the average coefficients, and the detail coefficients are stored in the remaining bands. The transform is recursively applied to the LLL band, generating new levels of subbands until we get four levels of transform. These levels are the basis of the multiresolution system.

To avoid any data loss during the wavelet application the coefficients are preserved without modifications. This means that the magnitude of the coefficients (specially the low-frequency ones) grows each time the transform is applied. This approach increases the storage requirements but guarantees that the only source of data loss is in the later quantization step.

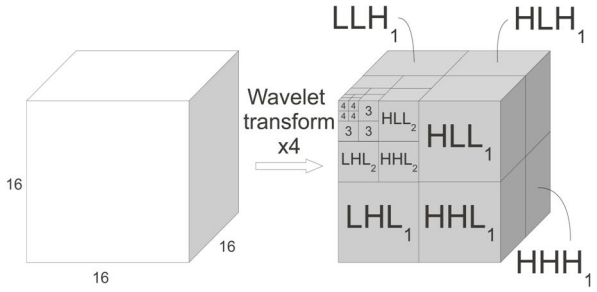


Fig. 3 – The result of applying a 4-level wavelet transform to a 16 × 16 × 16 block of data.

Quantization is a lossy compression technique that reduces the range of the values of the compressed dataset [18]. In our implementation we have chosen a scalar quantization solution with fixed-rate coding that removes the least significant bits of the coefficients obtained from the previous wavelet transform. This quantization reduces the magnitude of the coefficients, and nullifies those with a close-to-zero value. The quantization level must be decided according to the compression quality, where not only the compression ratio, but also the signal to noise ratio are considered.

The encoding step converts the resulting volumetric data from the wavelet-transform and quantization steps into its final compressed form following a compromise between good compression ratio and fast random access.

Fig. 4 shows the main data structures used in this step and their meaning. The cell-tag table array stores a cell-tag table for each block in the volume. A cell-tag table contains two-byte tags labeling each cell in a block. The most significant byte stores the width in bytes of the coefficients in the cell (or zero for a null cell), and the less significant byte stores the index of the significance map for the cell. The significance map array contains a bitmap for each non-null cell in the volume. This bitmap is used to flag coefficients in the cells as zero or nonzero. Although it has not been represented in Fig. 4, it is also necessary to store an offset value for each cell. It is stored in 8 bytes (long integer) and contains the corresponding non-null coefficient array.

Finally, four arrays store all the non-null coefficients from the transformed volume data. Each coefficient is stored in an array depending on its width in bytes. This encoding supports coefficients of up to four bytes.

Our encoding solution increases the flexibility of the implementation presented in [2], which was

limited to 4 × 4 × 4-cell blocks. Two-byte cell tags enable using bigger blocks, so more resolution levels could be supported, as the maximum number or recursive wavelet transforms that can be applied is restricted by the size of the block.

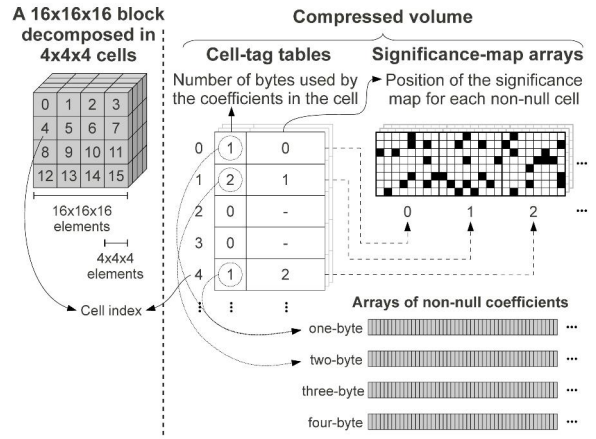


Fig. 4 – Structure of the encoded data.

3.2. Visualization

The visualization stage is responsible of reconstructing the compressed volume on the GPU (decoding and inverse wavelet transform computation) and rendering it on the screen. Fig. 5 shows the different steps that take place in this stage. The volume is processed brick by brick.

First, a brick is selected from the compressed volume and reconstructed at a specific level of resolution. This reconstruction involves the steps of decoding and inverse transform, which have been implemented in CUDA kernels, and hence, run on the GPU. The restored brick data are stored in an OpenGL Pixel Buffer Object (PBO), and then copied into a texture buffer to be mapped onto a proxy geometry. These operations including the final rasterization are implemented using the OpenGL API. The process continues with another brick until the complete volume has been rendered.

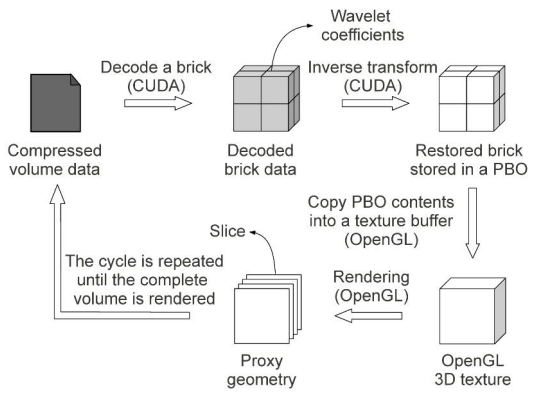


Fig. 5 – Visualization system for decompressing and rendering the volume data.

The visualization process is performed brick by brick. In each frame, the CPU decides in which order bricks should be reconstructed according to the position of the camera. A back-to-front order is maintained to guarantee a correct composition of the bricks.

For each brick, depending on its distance to the camera, the CPU chooses a resolution level. Bricks that are close to the camera are rendered at the highest level. Bricks that are far from the camera do not contribute to the final result as much as the closer ones, so in order to speed up the whole process they are rendered at a lower level of resolution.

As stated earlier, two CUDA kernels execute the steps of decoding and inverse transform required to decompress the brick data in order to reconstruct the whole volume. The decompressed data are stored in a PBO, which can be accessed by the CUDA and OpenGL functions.

To complete the visualization, an OpenGL call copies the brick data from the PBO into a texture buffer. Then, the CPU orders the construction of the proxy geometry using several OpenGL calls. This proxy geometry contains the slices where the brick texture is mapped onto.

Depending on the resolution level, the texture might not completely fill the available space in the texture buffer. That is, the highest resolution level uses the complete texture space, but low-resolution textures require only a small portion of that space. This means that the texture coordinates assigned to each vertex of the proxy geometry must be adjusted to the real texture size according to the resolution level chosen for the current brick.

Analyzing each step of the visualization stage more in detail, first we have to pay attention to the decoding step. The process of decoding is performed in a kernel on the GPU. This kernel reads the compressed data of the brick from the compressed volume stored in the GPU global memory and writes the decoded data in a previously allocated buffer (to be later processed by the inverse wavelet transform). Each data block in the brick is assigned to a thread block, where each thread processes a cell (whose size is 4 × 4 × 4 in our implementation).

The decoding process is as follows. Each thread starts by determining if its cell is non-null or not, as indicated by the cell tag associated to the cell. If the cell is non-null, the data reconstruction begins. The thread loops through the elements of the cell, and tries to load them from the arrays of non-null coefficients in the compressed volume (see Fig. 4) accessing the information stored in the significance-map of the cell and considering the offset value for the cell. If the cell's significance map identifies a

coefficient as non-null, its value is stored as is in the buffer in global memory, otherwise a zero is written.

For each brick, data from the different blocks are initially interleaved in global memory attending to their absolute position in the brick. In order to increase the spatial locality of memory accesses, the decoded data are contiguously stored in global memory in a blockwise fashion. Our proposal arranges the data of each block together to reduce the time spent in memory accesses when the inverse wavelet transform is computed.

Once the decoding kernel has finished, another kernel performs an inverse wavelet transform on the GPU to restore the brick contents. Each data block in the brick is assigned to a thread block depending on its identifier, and each thread processes a chunk of 2 × 2 × 2 voxels although this could be modified with minor changes in the implementation.

The inverse transform is a recursive process, and it is applied until the desired level of resolution is achieved. The resulting coefficients of processing a level of resolution are stored in shared memory, where this data will be available to compute the next level of resolution. When the desired level is reached, these coefficients are copied from shared memory into the OpenGL PBO. In the case of processing the highest level of resolution, the coefficients are directly stored in the PBO, bypassing the shared memory and consequently reducing its impact on the memory load.

When storing data in the PBO, the positions where data are placed are determined by the identifiers of the thread block and the current resolution level. For low resolution levels, the data generated by each thread block are grouped in order to avoid chunks of data scattered in the PBO.

Fig. 6 shows how a 32 × 32 × 32 restored brick is stored in the PBO for different levels of resolution.

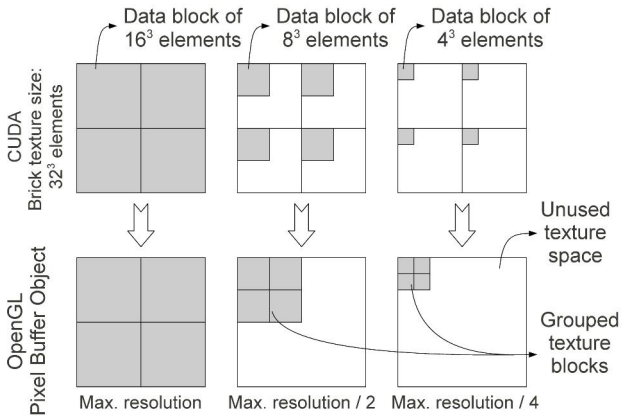


Fig. 6 – Storing data from shared memory into the PBO for different resolutions.

4. RESULTS

4.1. NUMERICAL RESULTS

We performed our tests on a machine consisting on a CPU multicore and a GPU. The CPU is an Intel Core 2 Quad Q9450 with four cores at 2.66 GHz and 6 GB of RAM. The GPU is a NVIDIA GeForce GTX 580 with 16 SMs of 32 SPs each featuring a total of 512 processor cores operating at a clock rate of 1.544 GHz, and with 1.5 GB of global memory. Each SM has 64 kB of RAM with a configurable partitioning of shared memory and L1 cache (16 kB of shared memory and 48 kB of L1 cache, or vice versa). Additionally, a unified L2 cache of 768 kB is available for all SMs [16].

We compiled the code using the NVIDIA nvcc compiler provided within the CUDA 4.0 toolkit and the gcc version 4.4.3 under Linux.

Table 1 details the different datasets used in this work that can be considered as representative instances of volumes obtained from organic tissues and synthetic materials The *BrainWeb* dataset was obtained at the BrainWeb Simulated Brain Database [19]. *ModelHead* corresponds to a volumetric CT of a synthetic model of the human head, whereas *RealHead* is volumetric dataset of a real human head obtained with an MRI technique. The last two volumes, A80 and Knee-001, were obtained through segmentation using a GPU-accelerated level-set segmentation algorithm on two datasets comprising contrast-enhanced CT images. In the case of A80 the dataset corresponds to several brain vessel images that presented some observable cases of aneurysms. Knee-001 corresponds to a knee image. Fig. 7 shows renderings from the datasets using our solution.

Table 1. Datasets used in this work considering that in all the cases the number of bytes per voxel is 2.

Name	Size	File Size
RealHead	160 × 512 × 512	80 MB
Brainweb	256 × 256 × 181	23 MB
ModelHead	512 × 512 × 348	174 MB
A80	512 × 512 × 512	256 MB
Knee-001	256 × 256 × 256	32 MB

4.2. QUALITY ANALYSIS AND STORAGE REQUIREMENTS

We have measured the quality of the proposed decoding implementation with the volumes described in Table 1. Tables 2 and 3 show the values obtained for the *BrainWeb* and *ModelHead* datasets for different levels of quantization.

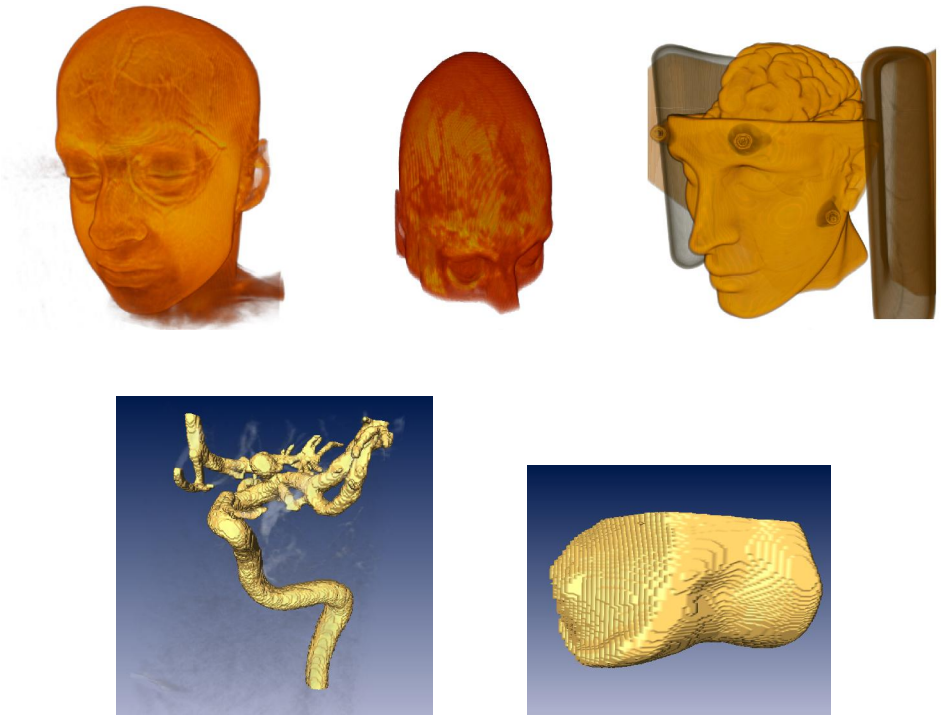


Fig. 7 – Volume rendering of the test datasets. From left to right and from top to bottom: *RealHead, BrainWeb*, *ModelHead, A80* and *Knee-001.*

Quality is measured here in terms of compression ratio, mean squared error (MSE) and peak signal-to-noise ratio (PSNR). A quantization level corresponds to removing a specific number of least significant bits from the coefficients of the wavelet transform (see Section 3.1.2). Eleven different quantization levels are considered in the experiments, as indicated in the table.

Generally, a value of PSNR above 60 is considered good, so we have chosen a quantization level of 8 bits in our tests to measure performance of the complete GPU volume rendering system (see below). We noticed that changing the number of bits removed during quantization did not significantly affect the performance measured in terms of execution times, however it does severely affect the storage requirements as it was explained in Section 4.1.

The compressed volume is stored in different data structures that were detailed in section 3.2 and shown in Fig. 4. Considering the implementation issues it can be concluded that the nine integer arrays that will be detailed in the next paragraphs are required in order to store the compressed volume.

The cell tags for the cells in the volume are stored in a 2-byte integer array. In addition, for each non-null cell a significance map is generated, so the indices of the maps, the maps themselves and the offsets to access the coefficients are stored in three 8-byte integer arrays.

Table 2. Compression quality for different levels of quantization for the BrainWeb dataset.

# bits	Compr. volume size (MB)	Compress. ratio	MSE	PSNR
0	25.52	1 : 0.89	0.00	∞
1	25.10	1 : 0.90	0.45	99.77
2	18.92	1 : 1.20	0.72	97.75
3	16.81	1 : 1.35	2.28	92.76
4	14.85	1 : 1.52	8.84	86.86
5	13.25	1 : 1.71	35.34	80.85
6	11.41	1 : 1.98	137.92	74.93
7	8.63	1 : 2.62	509.06	69.26
8	5.57	1 : 4.06	1614.11	64.25
9	2.89	1 : 7.83	3488.04	60.90
10	1.81	1 : 12.50	6628.75	58.12
11	1.15	1 : 19.67	12135.79	55.49

All the non-zero coefficients are stored in four byte arrays. These arrays contain as many bytes as required by the non-null coefficients of different lengths.

Finally one more 8-byte pointer array per brick is required in order to store the indices of the non-null blocks in the brick.

Table 3. Compression quality for different levels of quantization for the ModelHead dataset.

# bits	Compr. volume size (MB)	Compress. ratio	MSE	PSNR
0	111.82	1 : 1.56	0.00	∞
1	89.08	1 : 1.95	0.45	99.79
2	64.35	1 : 2.70	0.63	98.31
3	45.87	1 : 3.79	1.34	95.05
4	33.02	1 : 5.27	3.25	91.21
5	23.95	1 : 7.27	8.58	87.00
6	17.74	1 : 9.81	23.81	82.56
7	13.09	1 : 13.29	66.71	78.09
8	9.57	1 : 18.18	187.64	73.60
9	7.08	1 : 25.48	504.59	69.30
10	5.29	1 : 32.89	1320.25	65.12
11	3.82	1 : 45.55	3117.75	61.39

For the ModelHead image in Table 1 whose size is 512 × 512 × 348 voxels with 2 bytes/voxel, i.e. a volume of 174 MB, and considering a brick size of 128 × 128 × 128, 16 × 16 × 16 blocks and 4 × 4 × 4 cells, and a quantization level of 8 bits, the size required to store the compressed volume is 9.57 MB. This size could be reduced, as it can be observed in Table 3, if a higher level of quantization is selected, thus loosing quality in the visualized image.

4.3. PERFORMANCE ANALYSIS

In order to evaluate the performance we focus on the GPU implementation of the different steps of the rendering system. In particular, we have measured execution times and speedups of the decoding and inverse-transform kernels compared to the CPU implementations. Then, we executed the complete system on the GPU and took measures of execution time for each step and of FPS for the whole system varying the volume size and brick size parameters.

Regarding the speedup measurements, we focus on the implementations of the decoding and the inverse-transform steps implemented in CUDA. Table 4 shows the results we have obtained for different volume sizes that were constructed from the *RealHead* dataset and considering average values for only one brick. High speedups are obtained for both kernels, especially for the inverse transform. For both algorithms, the speedup increases with the volume size, as the computational capabilities of the GPU are better exploited when the number of working threads is larger.

We also evaluate the performance of the whole visualization process on GPU showing the execution times for each step and the frames per second (FPS) obtained. Table 5 shows the performance for two of the datasets whose sizes are described in Table 1 using different brick sizes. As for the other experiments the total time per brick is calculated and multiplied by the number of non-null bricks (the null ones do not require computations) obtaining the total time ("Total" in the Tables). The FPS value is directly calculated from this value. From Table 1, we see that, in general, the larger the brick size, the better the performance obtained. Generally, incrementing the brick size increases the time required to process a brick, but reduces the number of bricks, resulting in a lower time to complete a frame.

Table 6 details results for the *A80* and *Knee-001* datasets. In these cases there are empty bricks in the volumes, i.e. bricks that do not require computation. So, the number of non-null bricks are also specified in the table and considered in the computation of the time per frame ("Total" in the Table). As in Table 5, for the same volume bigger bricks sizes require smaller number of bricks and, therefore, smaller execution times and higher FPS rates.

Table 4. Execution times in seconds and speedups respect to the CPU implementation of the decoding and inverse-transform kernels operating on a single brick of the *RealHead* image for different brick sizes.

Kernel		64 × 64 × 64	128 × 128 × 128	256 × 256 × 256
Decoding	GPU	0.000077	0.000196	0.001205
	CPU	0.003173	0.024237	0.207878
	Speedup	41.2x	123.7x	172.5x
Inverse Transf.	GPU	0.000027	0.000192	0.001526
	CPU	0.009205	0.069018	0.557935
	Speedup	340.0x	352.6x	365.6x

Table 5. Execution times (seconds) and calculated FPS for the steps of the GPU rendering system varying the brick size and considering the *RealHead*, *BrainWeb* and *ModelHead* volumes. All the bricks are non-null.

Dataset	Brick Size	# of bricks	Decode per brick (CUDA)	Inv. Transf. per brick (CUDA)	Copy per brick (OpenGL)	Render per brick (OpenGL)	Total per brick	Total	FPS
RealHead	64^3	192	0.000077	0.000027	0.000016	0.000382	0.000502	0.0963	10
	128^3	32	0.000196	0.000192	0.000023	0.000699	0.001110	0.0355	28
	256^3	4	0.001205	0.001526	0.000038	0.001546	0.004315	0.0172	57
BrainWeb	64^3	48	0.000081	0.000027	0.000016	0.000726	0.000850	0.0408	24
	128^3	8	0.000226	0.000191	0.000023	0.001188	0.001628	0.0132	75
	256^3	1	0.001571	0.001534	0.000038	0.002282	0.005425	0.0054	184
ModelHead	64^3	384	0.000075	0.000027	0.000016	0.000529	0.000647	0.2484	4
	128^3	48	0.000215	0.000192	0.000023	0.000940	0.001369	0.0657	15
	256^3	8	0.001206	0.001524	0.000037	0.001559	0.004326	0.0346	29

Table 6. Execution times (second)s and calculated FPS for the steps of the GPU rendering system using *A80* and *Knee-001* datasets and varying the brick size. The number of non-null bricks is also specified.

Dataset	Brick Size	# of bricks/ non-null bricks	Decode per brick (CUDA)	Inv. Transf. per brick (CUDA)	Copy per brick (OpenGL)	Render per brick (OpenGL)	Total per brick	Total	FPS
A80	32^3	4096/531	0.000062	0.000011	0.000013	0.000246	0.000332	0.176	5
	64^3	512/129	0.000055	0.000009	0.000016	0.000451	0.000531	0.068	14
	128^3	64/31	0.000060	0.000023	0.000023	0.000819	0.000925	0.029	34
	256^3	8/7	0.000139	0.000088	0.000037	0.001565	0.001829	0.013	76
	512^3	1/1	0.000819	0.000564	0.000061	0.002624	0.004068	0.004	250
Knee-001	32^3	512/115	0.000060	0.000012	0.000013	0.000431	0.000516	0.059	16
	64^3	64/29	0.000048	0.000013	0.000017	0.000806	0.000884	0.026	38
	128^3	8/8	0.000058	0.000030	0.000023	0.001531	0.001642	0.013	76
	256^3	1/1	0.000240	0.000226	0.000037	0.002981	0.003484	0.003	333

4.4. COMPARISON TO OTHER WORKS

To the best of our knowledge, this is the first GPU implementation of a decompression scheme based on [2].

The authors reported their solution required, at best, nearly 10 seconds to reconstruct a volume of 512 × 512 × 512 elements on CPU. This includes both the decoding step and the inverse transform step. For a brick of the same size, Table 4 shows a performance between 15 and 20 milliseconds for both steps on the GPU.

Our inverse wavelet transform compares favorably with other GPU implementations in the literature. In a recent work [20], the performance of a 3D fast wavelet transform was measured on a GPU processing 64 frames of a video at different resolutions, requiring 6.8 ms for a 512 × 512 video, and 13.4 ms for a 1024 × 1024 video. This implementation performed a one-level transform using a Daubechies D4 wavelet [21]. To compare these results, we have measured the performance of our inverse-transform kernel for a single level instead of four. Processing a brick of size 256 × 256 × 256, which is exactly the same size as the former video, requires 1 ms in our solution. A 512 × 512 × 512 brick, which is twice the size of the latter video, requires 7 ms.

The performance of the GPU decompression and rendering system is also competitive with similar solutions in the literature. A scheme based on the Karhunen-Loève transform [22] is presented in [1]. Compression is performed on CPU using a vector quantization approach that preserves the coefficients from blocks containing the most relevant edges. Visualization is achieved in a two-pass render, the first one devoted to decompress several slices of data, and the second one to the actual rendering. A 512 × 512 × 512 is rendered at a rate between 6 and 11 FPS, depending on the size of the viewport. For a volume with a similar size (*ModelHead*), our solution achieves 29 FPS without the size of the viewport affecting significantly.

Finally, a solution based on the S3 texture compression algorithm (also known as DXT) [23] was introduced in [24] for time-varying 3D datasets. The reconstruction of the compressed volume data is embedded into a programmable shader, and up to three frames are compressed into the RGB channels of a texture. The authors show results for a volume of size 400 × 600 × 400 visualized at 35 FPS. Although this performance is slightly higher than our solution's, our compression scheme provides better results in terms of quality, with a greater PSNR for a similar compression ratio.

5. CONCLUSIONS

In this work we have presented a GPU solution for decompressing and visualizing 3D datasets using a multiresolution rendering scheme. A previous compression stage based on wavelets, and performed in the CPU, is required. The selection of the quantization level applied to the wavelet coefficients is the factor that decides the compression rate and the SPNR value of the compressed volume. A tradeoff value of 8 bits is selected for the quantization level.

The GPU stores the compressed version of the original volume. Our GPU solution processes the compressed volume in 3D data pieces called bricks. For each brick a level of resolution is selected depending on its distance to the camera, and the brick data are decompressed up to that level.

The decompression involves decoding and computing the inverse wavelet transform of the data. Both steps are implemented in CUDA, so they are executed within the GPU. Unlike other out-of-core techniques, communication between CPU and GPU is minimal, avoiding the bottleneck that the PCI bus between both is. As we apply four levels of wavelet, our approach supports up to four different levels of resolution (five including the original one).

The visualization is carried out using the texture mapping technique. The decompressed brick data is copied into an OpenGL texture buffer and mapped onto a proxy geometry composed of several parallel polygonal slices. The GPU rasterizes the geometry by blending the slices to produce the final image.

The solution has been tested with five medical datasets obtaining competitive results compared to other recent GPU implementations of compressed volume rendering. The refresh rates obtained are competitive, the PSNR values are greater than 60, and a compression ratio between 1:4 and 1:18 for volume sizes in the range between 64^3 and 256^3 is obtained. A higher quantization level, that could give enough quality for some applications, would increase the compression rate of the solution at the cost of worsening these quality parameters.

As future work, we plan to extend our solution to larger datasets, including datasets that do not fit inside the GPU memory. For these cases, empty-space–skipping techniques are essential to identify bricks in the volume that do not add essential information to the final rendering in order to keep an interactive refresh rate.

6. ACKNOWLEDGEMENTS

This work was supported in part by the Ministry of Science and Innovation, Government of Spain, and FEDER funds under contract TIN 2010-17541, and by the Xunta de Galicia under contracts 08TIC001206PR and 2010/28. Julián Lamas-Rodríguez acknowledges financial support from the Ministry of Science and Innovation, Government of Spain, under a MICINN-FPI grant.

7. REFERENCES

[1] N. Fout and K.-L. Ma. Transform coding for hardware-acceleratedvolume rendering, *IEEE Transactions on Visualization and Computer Graphics*, (13) 6 (2007), pp. 1600-1607.

[2] I. Ihm and S. Park, Wavelet-based 3D compression scheme for interactive visualization of very large volume data, *Computer Graphics Forum*, Lake Tahoe, CA, (18) 1 (May 2-5, 1999), pp. 3-15.

[3] S. Guthe, M. Wand, J. Gonser, and W. Straßer, Interactive rendering of large volume data sets, in *IEEE Visualization 2002*, Boston, Massachusetts, (Oct. 27-Nov. 1, 2002), pp. 53-60.

[4] J. Schneider and R. Westermann, Compression domain volume rendering, in *IEEE Visualization 2003*, Seattle, Washington, (Oct. 19-24, 2003) pp. 293-300.

[5] R. Parys and G. Knittel, Giga-voxel rendering from compressed data on a display wall, *Journal of WSCG*, (17) 13 (2009), pp. 73-80.

[6] E. Gobbetti, J. A. Iglesias Guitián, and F. Marton, COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks, *Computer Graphics Forum*, (31) 3-4 (2012), pp. 1315-1324.

[7] S. K. Suter, J. A. Iglesias Guitián, F. Marton, M. Agus, A. Elsener, C. P. Zollikofer, M. Gopi, E. Gobbetti, and R. Pajarola. Interactive multiscale tensor reconstruction for multiresolution volume visualization, *IEEE Transactions on Visualization and Computer Graphics*, (17) 12 (2011), pp. 2135-2143.

[8] M. B. Rodríguez, E. Gobbetti, J. I. Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. Suter, A survey of compressed GPU-based direct volume rendering," *Eurographics 2013,* Girona, Spain, *(May 6-10, 2013)*, pp. 117-136.

[9] Julián Lamas-Rodríguez, Francisco Argüello, Dora B. Heras, "A GPU-based Multiresolution Pipeline for Compressed Volume Rendering", *The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, EEUU, (July 22-25, 2013), pp. 523-529.

[10] A.V. Gelder and K. Kim, Direct volume rendering with shading via three-dimensional textures, *1996 Symposium on Volume Visualization*, (Oct. 28-29, 1996), pp. 23-30.

[11] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf, *Real-time volume graphics*. A K Peters, Ltd., 2006.

[12] Q. Zhang, R. Eagleson, and T. M. Peters. Volume visualization: a technical overview with a focus on medical applications, *Journal of Digital Imaging*, (24) 4 (2011), pp. 640-664.

[13] E. Gobbetti, F. Marton, and J. A. I. Guitián. A single-pass GPU raycasting framework for interactive out-of-core rendering of massive volumetric datasets, *The Visual Computer,* (24) 7–9, (2008), pp. 797-806.

[14] B. Liu, G. J. Clapworthy, F. Dong, and E. C. Prakash. Octree rasterization: accelerating high-quality out-of-core GPU volume rendering, *IEEE Transactions on Visualization and Computer Graphics*, (99) (2012), pp. 1-14.

[15] J. Nickolls and W. J. Dally. The GPU computing era, *IEEE Micro*, (30) 2 (2010), pp. 56-69.

[16] *CUDA C programming guide (version 4.0), NVIDIA, 2011.*

[17] D. B. Kirk and W.-m. W. Hwu, *Programming massively parallel processors: a hands-on approach.* Burlington, Massachusetts, USA: Elsevier, 2010.

[18] R. M. Gray and D. L. Neuhoff, Quantization, *IEEE Transactions on Information Theory*, (44) 6 (1998), pp. 2325–2383.

[19] C. Cocosco, V. Kollokian, R.-S. Kwan, and A. Evans, BrainWeb: online interface to a 3D MRI simulated brain database, *NeuroImage*, (5) 4 (1997), p. S425.

[20] G. Bernabé, G. D. Guerrero, and J. Fernández, CUDA and OpenCL implementations of 3D fast wavelet transform, *3rd IEEE Latin American Symposium on Circuits and Systems*, Playa del Carmen, Mexico, (Feb. 29- March 2, 2012), pp. 1-4.

[21] I. Daubechies, *Ten Lectures on Wavelets*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1992.

[22] R. D. Dony, *Karhunen-Loève transform*, in The Transform and Data Compression Handbook. Boca Ratón, Florida, CRC Press, 2004.

[23] K. I. Iourcha, K. S. Nayak, and Z. Hong, *System and method for fixed-rate block-based image compression with inferred pixel values*, US Patent 5 956 431, Sept. 21, 1999.

[24] Y. Cao, L. Xiao, and H. Wang, "Hardware-accelerated volume rendering based on DXT compressed datasets," *International Conference on Audio, Language and Image Processing*, Shangai, China, (Nov. 23-25, 2010), pp. 523-52.

Julián Lamas Rodríguez, *received his B.Sc in Computer Engineering from the University of Coruña, Spain, and his M.Sc in Videogame Creation from the University Pompeu-Fabra in Barcelona, Spain, both in 2006. In 2007 he joined the Systems Laboratory Group in the Technological Research Institute of University of Santiago de Compostela. In 2009 he joined the Computer Architecture Group of the Department of Electronics and Computer Science in the same University. He recently joined the Visualization and Data Analysis Department at Zuse-Institut Berlin. The scope of his research is centered in high performance computing using graphics processors.*

Francisco Argüello Pedreira, *received the B.S. and Ph.D. degrees in Physics from the University of Santiago, Spain in 1988 and 1992, respectively. He is currently an Associate Professor in the Department of Electronic and Computer Engineering at the University of Santiago de Compostela, Spain. His current research interests include signal and image processing, computer graphics, parallel and distributed computing, and quantum computing.*

Dora Blanco Heras*, received a M.S. degree in Physics in 1994 and a Ph.D. in 2000 from the University of Santiago de Compostela (Spain). She is currently an Associate Professor in the Department of Electronics and Computer Engineering at the same University. Her research is in the field of parallel and distributed computing, including, for example, software optimization techniques for emerging architectures, and on computer graphics and image processing.*

computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

TOOLS FOR BIOMEDICAL DATA ARCHIVING IN UKRAINIAN GRID INFRASTRUCTURE

Oleksandr Sudakov, Andrii Salnikov, Ievgen Sliusar, Oleksandr Boretskyi

Information and Computer Centre, National Taras Shevchenko University of Kyiv,
Ukraine, Kyiv, 4D Glushkova prosp., e-mail: cluster@cluster.kiev.ua, http://grid.org.ua

Abstract: Tools for archiving and extraction of data in Ukrainian National Grid for end-users' applications are proposed, implemented and deployed for practical applications in medical imaging, non-linear dynamics, and molecular biology. Proposed tools provide the facilities to utilize large distributed storage space in grid infrastructures for different practical tasks including desktop applications. Tools may be successfully used even when on client platforms it is impossible to setup grid middleware, use web browser interfaces or grid security infrastructure authentication. Tools consist of extensible client compatible with different software and hardware platforms; web service for data transfer; web service for transparent data replication on grid storage elements.

Keywords: Grid, Replication, Web service, Client, Storage, Protocol, Graphics, Anonymization, Image, DICOM.

1. INTRODUCTION

Computing Grids [1] are geographically distributed means for high-throughput computing and large data storage in scientific, industrial, commercial and others branches of activity. Ukrainian National Grid (UNG) infrastructure [2] was created in 2006 and is being used in number of applied projects like molecular dynamics simulations [3], neuroscience [4], non-linear dynamics [5] etc. In all these applications large amount of data is generated by imaging devices, simulation or data processing software. Grid infrastructure provides consolidated capabilities for data storage, high performance computing, data replication, user interfaces and other services. Grid also provides means for data security, reliability and high throughput access. The idea to transparently use large distributed grid storage space to store data from end-user server and desktop applications is very attractive because it eliminates the necessity to create big data centers at the user side, reduces costs for administration etc. Such transparent grid storage system has features of cloud or big-data storages and should be implemented using grid services.

The most common way to access grid resources is setup of grid middleware on client platform, via web interfaces or accessing third party grid access platforms. Unfortunately in many cases transparent usage of grid infrastructure from desktops, servers and devices is not easily possible. For example, usually it is not possible to setup grid middleware on the medical imaging device hardware, web interface is no usually good to transfer large amount of data, it is not always possible to provide personal certificates for all possible users etc. Many tasks may have other special requirements.

In present work we introduce a set of tools for working with data in grid infrastructure using conventional web services and extensible clients: RAPTOR replication and data exchange service, web-interface for AMGA metadata catalogue and grid-enabled DICOM client. These tools may be easily used for application of grid infrastructure to work with medical and other data on workstations without grid middleware on large number of hardware and software platforms. Proposed tools proved to be rather universal and efficient for wide number of application including medicine, dynamical simulations and molecular biology.

2. UNG GRID DATA STORAGE REQUIREMENTS

The most specific requirements for grid storages in UNG arise from medical applications [6]. For a last few years several projects concerned with medical applications of UNG have arisen. These projects are held in the virtual organizations (VO) [7] medgrid [8], telemed and others and are devoted to sampling, archiving, reconstruction, recognition, fusion etc. of electrocardiography

(ECG), electroencephalography (EEG), nuclear medicine, ultrasound (USI) and other modalities data. Grid infrastructure provides consolidated capabilities for data storage, high performance computing, data replication, user interfaces and other services.

Data-intensive and computation-intensive medical applications have some special features. 1) Medical data is usually being transferred in special formats, like DICOM (Digital and Image Communication in Medicine) [9]. 2) Medical images contain personal patients' information and transfer of these data without patients' consent is legally prohibited. So it should be impossible to recover patients' personal data from images stored in the grid infrastructure for persons who are not authorized. 3) Medical images series may have large size up to about 1 GByte per examination and its transfer may require relatively large time and evidently may be aborted. 4) Medical images usually have large pixel depth (12-16 bits), may contain additional data and thus require special visualization techniques not available in conventional web browsers. 5) Jobs processing in grid usually introduce high latencies so client software should take this into account. 6) Credential management procedures for grid environment like personal grid certificates enrollment and renewal are relatively complicated and not usually acceptable for patients and physician thus additional simplified authentication methods should be used.

There are implementations of grid tools for working with DICOM images, like MDM (Medical Data Management) [10]. All these tools assume that grid middleware is installed on the client machine or there is DICOM PACS server available for physician. Unfortunately most Ukrainian medical centers still lack PACS servers and DICOM images are created on physician workstation form other formats. It is impossible to install and configure gLite middleware on client MS Windows-based workstations and there are many other reasons why MDM or other grid DICOM tools is hardly to be used now.

Other UNG grid applications have not so strict requirements and tools designed for medical data may be easily adapted to them.

3. MEDICAL DATA STORAGES IN UNG

Taking into account all the requirements stated above the following medical data storage approach was suggested for UNG. All data is physically stored on the replicated storage elements (SE). Replica locations are managed by highly available LFC (LCG File Catalog) grid services; data access to SEs is controlled by SRM (Storage Resource Manager) [11] grid services; clients accesses are performed by specially designed web services that interact with SRM. SRM standard provides implementation-independent interface to storage management functions e.g. third-party data transfers, storage space tokens and reservations, nearline storage interaction and access control. All data on SEs is stored in anonymized form. It means that all patients' personal information should be wiped from the metadata on clients before it gets transferred to grid. All data on grid storages may be accessed by authorized VOs members and may be used for processing on grid computing elements (CE).

A decision was made that all the Storage Elements which are devoted to medical data in the UNG should support SRM control protocol. Currently, all EGI-certified SE implementations (Disk Pool Manager, dCache and StoRM) support SRM which also ensures compatibility with various client tools. Currently, not all resource centers supporting medical image processing VOs got certified for operation in the EGI and hence information about storage resources is not published in the global resource indexes.

Dedicated information index was set up within the medgrid VO which collects information from all relevant SEs as well as LFC services. This allowed implementing web services for storage interaction relying on standard EMI client tools and libraries for data management.

Access to the SEs is controlled on the VO membership basis. Permissions to access certain files are deduced from VOMS [7] groups and roles that get assigned to users and services by the VO managers. A highly available VOMS service ensures data access permissions integrity because all the Storage Elements are configured to download corresponding information from VOMS.

To facilitate data exchange between medical information systems and portals and the grid storage infrastructure, web services were developed and deployed in the UNG.

The file exchange web-service (GW-FEX) [12] in Fig. 1 provides a gateway for exchanging data between web and grid infrastructure. The service is implemented as asynchronous request processor and provides RESTful interface for submitting new data transfer requests and querying status of previously submitted ones. Representational State Transfer (REST) [13] is a modern architectural concept of interactive multimedia (hypermedia) distributed systems.

From the web-site, the service is able to download and upload files to FTP and WebDAV (Web Distributed Authoring and Versioning) [14] shares. In the common usage scenario, web-portal has a spool folder shared by one of the supported

protocols accessible from the gateway service host. Requests are initiated by the web-portal. After successful completion of put-to-grid request, the gateway service returns grid identifier of the stored file which can be used later to submit pull-from-grid request for getting data back to the web-portal. Requests can be grouped to be processed in a batch. From the grid-side, the service operates with its own service certificate which is used to get authorization from VOMS and to access LFC data catalogs and SRM Storage Elements.

Permissions for a particular gateway are controlled by VOMS attributes assigned to the service. Standard EMI data management tools are used to implement data transfers and metadata registration in the grid infrastructure.

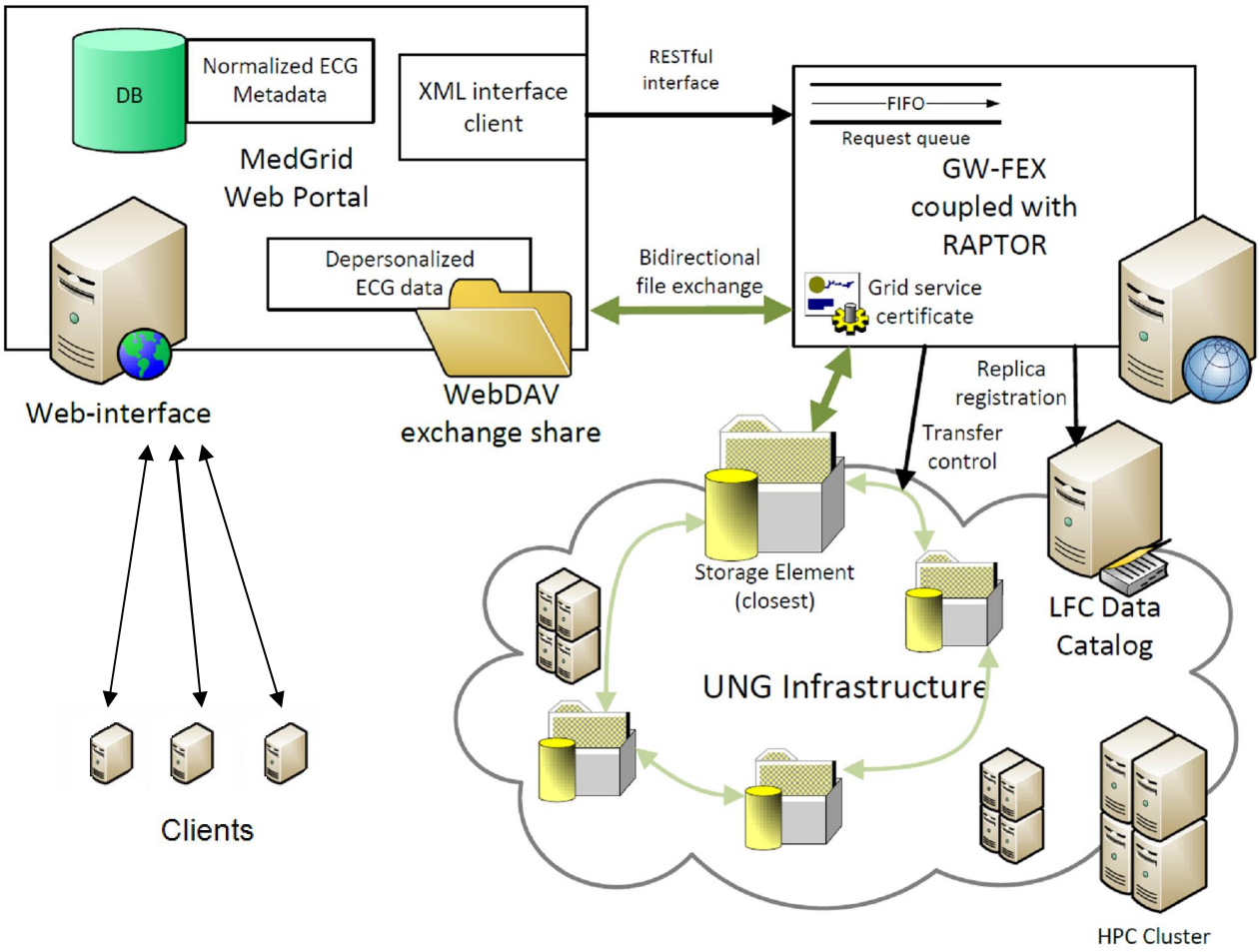


Fig. 1 – Web-service diagram.

Automatic data replication for maintaining high availability of data in the grid infrastructure is achieved by complementing the GW-FEX service with Robot for Autonomous Precisely Tunable Operation of Replication (RAPTOR) service [15]. RAPTOR service shares the request queue with the gateway service and continuously crawls through LFC catalog to verify replication policies for data stored in the grid. If some file was declared to be highly available with minimal number of replicas defined in the policy, then RAPTOR verifies availability of existing replicas and submits requests to produce new replicas when needed. This ensures that some particular file in the catalog minimally has a particular number of online replicas.

Consolidated deployment of GW-FEX and RAPTOR services provides a simple web service interface to reliable storage infrastructure for web-portals and hiding all the complexity of grid infrastructure internal operations. Consolidated installation of both services was employed by medgrid VO web-portal [7] to facilitate storage of ECG data.

The whole picture of interaction between web services and grid services is shown in Fig. 1. File exchange service places new data received from web portal on the closest grid storage element which is located on the same site. After successful upload, an entry for a file is created in the LFC data catalog and grid unique ID (GUID) gets assigned to the file.

From now, the file can be identified and accessed from the grid services.

Data catalog record eventually gets checked for conformance to a replication policy by the replication service. If having a single replica is not enough, requests for creating copies of the file on different SEs are pushed to the request queue. After successful replication, an entry in the LFC catalog is updated and new data locations are populated in the file record.

The RAPTOR service not just verifies a replicas but also checks its health. In the case of faulty SE data gets one more replica on a healthy one.

4. AMGA SERVICE

The ARDA Metadata Grid Application project (AMGA) [16] has emerged as a general purpose metadata catalogue service which allows attaching any relationally organized information to files stored in the grid infrastructure. AMGA can also be used as independent grid-enabled RDBMS service.

For medicine grid-enabled applications, AMGA is used for storing metadata found in DICOM images as well as electrocardiograms and other specific formats. Ability to select a dataset based on some properties like patient age, living location etc. is critical for population studies.

Besides providing a low-level API for database access, AMGA features a set of components for building web-based interfaces to metadata databases. This functionality had used in several projects across UNG that rely on central AMGA catalogue instance coupled with central grid file catalogue.

5. CLIENTS

Clients are specially designed to be compatible with different operating systems, to be independent of grid middleware and to be easily extensible. Portability is achieved by employment of portable software and libraries: GUI was implemented in Microsoft .NET; network components are implemented using open source software, like cURL library [17] and GNU compiles; database access is implemented with ODBC interface; DICOM parsing is provided in the portable DICOM Toolkit library [18].

User client in Fig. 2 consists of several modules interacting with each other and implemented as different processes or threads. Number of modules may be different depending task to solve. For medical application the main module is graphical user interface (GUI). It performs visualization of images in DICOM format. Visualizing module supports images with more then 8-bits pixel depth, manual and automatic changing of brightness scaling window position and width, multi-frame mode with animations. GUI also provides means to work with images data, open and save examinations, lookup in local database, implements functions associated with data transfer to/from grid infrastructure. New functions may be added by external software modules. Data exchange with grid infrastructure is performed by network agent. Some functions of DICOM data parsing are performed by DICOM utility. Data anonymizing/deanonymizing is performed by anonymizer utility.

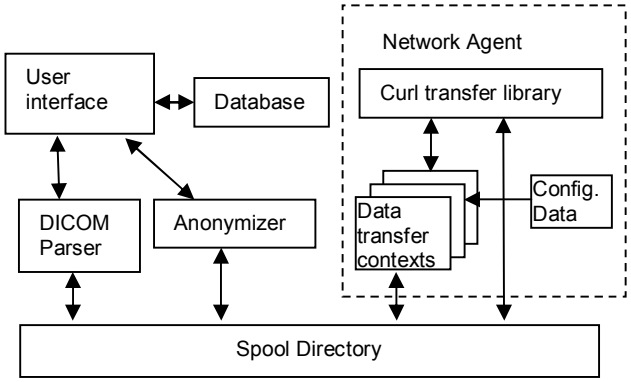


Fig. 2 – Client components diagram.

Interaction between modules is performed using spool directories on file-system and local database. Interactions via spool directories are used for transactions that have no references to other transactions or operate with data that should not be saved permanently. Data consistency in spool directories is provided by using simple request-response locking technique. Consumer creates temporary request file and adds information to it. When finished temporary request file is copied to main request file and after that temporary file is deleted. Provider periodically or by notification checks spool directory for main request files that have no temporary files. While request is executed its request file is used as log and transaction database. Information is added into request file when necessary. When execution is complete its request file is copied to response file and request file is deleted. Consumer checks spool directory for response files without request files, acquires request processing results and deletes the response file. Such scheme being correctly used provides data consistency even in the cases of software or hardware failures, network transfer aborts etc. This exchange method does not require any additional libraries and is very portable.

Interactions via database are used for transactions that may be simultaneously accessed by different modules, when some records should be saved permanently or when search capabilities are necessary.

Anonymizer is an external software module that provides removal of patient's personal data from DICOM images before transferring data to the grid for storage and restoration of personal data after transferring images from the grid. Anonynizer is called by GUI for all images to be transferred.

For each image transferred to the grid anonymizer inserts patient ID into local database if it does not exist there and reads patient alias from database. This alias is written into anonymized DICOM file. Then values of DICOM tags needed to be anonymized are cleared from anonymized file and whole DICOM header of original file gets saved in the local database. Request to send anonymized file is passed to network agent. After transfer to the grid network agent returns response file to user interface which deletes all temporary data files and inserts grid file location URL into the local database. When images are transferred back from the grid storage, anonymizer restores the whole DICOM header from the database using patient alias. Image files that have beem transferred to the grid may be deleted on the client and restored later from the grid. Tags that should be anonymized are configured via client GUI and may be changed.

This approach provides complete anonymizing of data on client side and thus eliminates transfer of personal data. Personal information may be restored only on the client that performed anonymizing. Shortcoming of this approach is treatment of the same patient on different clients as different persons. This problem may be solved by exchanging of local database content between authorized clients.

Network agent provides interaction with grid and web services like data transfers, job submissions, search queries etc. It is implemented as multi-protocol daemon process that interacts with GUI and other client components via spool directory. Network agent is object oriented so new protocols may be easily added by redefinition of some callback functions in job context classes.

Network agent runs continuous loop where all job contexts' instances are executed. Now two job contexts concerned with data exchange with the grid infrastructures are implemented. Multiple instances of each job context are allowed to run in parallel so many data transfers to and from the grid may be performed simultaneously. Network transfer management is performed by cURL library. Each job context passes authorization to web service using login/password or personal user certificate. Then data is transferred via HTTP POST request. After transfer is complete, the response a file is passed to the GUI. This file contains status information, transfer statistics and grid URL of file for send request.

6. MEDICAL APPLICATIONS

The proposed clients are used to implement tools (Fig. 3) for archiving of medical images from different Healthcare institutions in Ukraine. The main goal of the project is creation of large archive of anonymized DICOM images of different modalities. One of such archives is created today by The Institute for Scintillation Materials National Academy of Sciences of Ukraine to store scintigraphic and SPECT images from gamma-cameras in different nuclear medicine institutions of Ukraine.

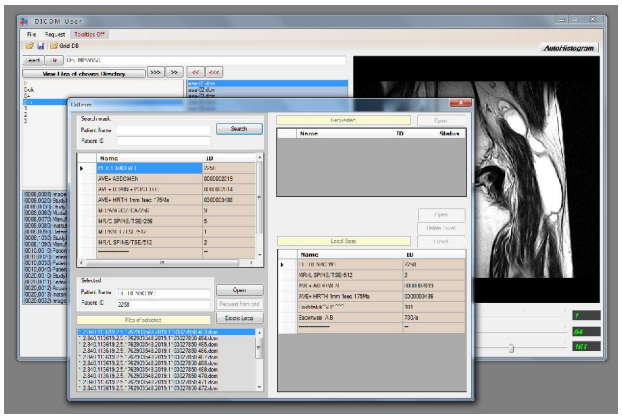


Fig. 3 – Medical application GUI screenshot

Direct usage of MDM or other grid DICOM tools for such tasks requires installation of grid middleware on workstations or access to PACS (Picture Archiving and Communication System) DICOM servers from grid sites. The first task is complicated and the second task is not usually possible because LANs of medical institutions are usually separated from the Internet. To simplify the task of the grid archive access, DICOM images are transferred to the grid infrastructure from physicians' workstations that can access local DICOM images and Internet access via firewall or proxy server by means of proposed clients.

Now there are three large projects in UNG devoted to medical images archiving. The first one is creation of emission tomography images archive in the Institute for Scintillation Materials National Academy of Sciences of Ukraine. This archive runs web-service in the Institute for Cybernetics National Academy of Sciences of Ukraine and has replicated storages in UNG. Kiev Municipal Heart Centre and Amosov National Institute of Cardio-pulmonary Surgery use this archive for reliable medical images storing.

Another project that use the services developed is creation of electrocardiograms (ECG) archive in the Institute for Problems of Mathematical Machines and systems National Academy of Sciences of

Ukraine. This archive is used in several medical institutions of Ukraine supporting medgrid VO [8].

Currently there are about 20 thousands of ECGs accumulated in the archive. Single ECG file has nearly a few megabytes in size and hence there was implemented aggressive data replication policy that specify creation of replicas of all ECGs on all accessible grid-storages [12].

Such approach enables for meeting all needed requirements for carrying out population-based studies on top of geographically distributed grid-sites [19].

The third project is creation of electroencephalograms (EEG) archive in UNG by the Bogomolets Institute of Physiology National Academy of Sciences of Ukraine. This archive is used in National Taras Shevchenko University of Kyiv, Ukraine and in The University of Vermont, USA for archiving human and rat EEG data for scientific research. This project also has the aim to create multimodal database and knowledge dataset of physiological data.

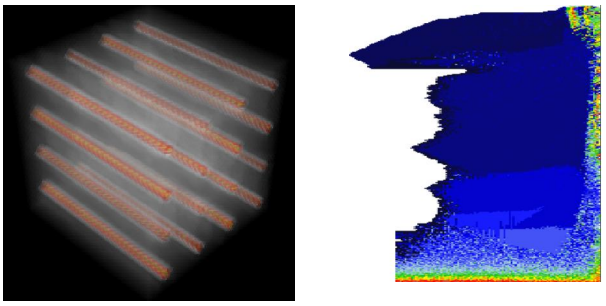
Another intended application of this client software is employment of grid computing resources for computation-intensive image reconstruction [20], fusion, recognition etc. There plans to create differential diagnostic service based on SPECT images, and USI images [21], procession of ECG and EEG data etc.

Now described clients lack capabilities for searching data in the grid. This function is planned to be implemented by interfacing AMGA Metadata Catalog grid services and integration with MDM in more distant future.

7. APPLICATIONS FOR MASSIVE SIMULATIONS

Proposed storage services also used for creation of massive scientific simulations results databanks. The first databank was created in the National Scientific Center for Medical and Biotechnical Research National Academy of Science of Ukraine [4, 5]. This data archive contains results of nonlinear dynamics simulations for different systems including neurons systems.

This archive contains about 400000 dynamics trajectories for 1d, 2d and 3d systems (Fig. 4a) simulated in UNG. Aggregation diagrams Fig. 4b are built from data in these archives. Each pixel of such diagrams corresponds to one trajectory in different parameters space. Clicking in appropriate place at the aggregation diagram gives a page with extended parameters of trajectory. Besides image representation diagram also is represented as table with sorting capabilities. Searching is possible on web page using web browser search functions.



a) b)

Fig. 4 – Archive of non-linear dynamics data: result of 3d system simulation (a), aggregation diagram (b).

One of the most successful UNG applications in the VO MolDynGrid [3] uses the replication services developed for storing simulation results and initial parameters of molecular dynamics in grid infrastructure for further complex analysis.

Currently there are about one thousand trajectories of the molecular dynamics. Because of huge size of one trajectory (usually from 10 GB to 100 GB for typical biological objects studies in virtual laboratory), a decision was taken to have from 2 to 3 online replicas of it on the grid storages available for VO.

Using balanced policy in the replication scheme allowed providing enough level of data redundancy, reducing load on grid storage elements and network communications.

8. CONCLUSION

Numerous successful applications of proposed grid-services and clients for archiving of medical images, EEG and ECG data, dynamical simulation results prove that presented tools provide means for quick creation and usage of grid archives. Tools operate directly from users workstations and clusters possibly eliminating complicated procedures of grid middleware installation, certificate obtaining etc. Built-in support for high performance data transfers, data anonymization and extensibility are distinctive features of this software suite. Future plans include more tight integration of proposed storage services with computing and data search facilities and creation of grid expert systems on using proposed storage services.

9. ACKNOWLEDGMENTS

Funding for the project was provided within The Ukrainian State Scientific and Technical Programme on Development and Application of Grid Technologies in 2009–2013 by State Agency on Science, Innovations and Informatization of Ukraine and National Academy of Sciences of Ukraine. All

research and development was held in Ukrainian National Grid (UNG) infrastructure on clusters of Information and Computer Centre National Taras Shevchenko University of Kyiv and National Scientific Centre for Medical and Biotechnical Research National Academy of Sciences, Ukraine.

10. REFERENCES

[1] I. Foster and C.Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.

[2] M. Zynovyev, S. Svistunov, Y. Boyko, and O. Sudakov, Ukrainian grid infrastructure: Practical experience, *Proceedings of the 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications IDAACS'2007*, September 6-8, 2007, Dortmund, Germany, pp. 165–169.

[3] A. O. Salnikov, I. A. Sliusar, O. O. Sudakov, O. V. Savytskyi, A. I. Kornelyuk, MolDynGrid virtual laboratory as a part of Ukrainian Academic Grid infrastructure, *Proceedings of the 5th IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications IDAACS'2009*, Rende (Cosenza), Italy, September 21-23, 2009, pp. 237-240.

[4] A. Salnikov, R. Levchenko, O. Sudakov, Integrated grid environment for massive distributed computing in neuroscience, *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications IDAACS'2011*, Prague, Czech Republic, September 15-17, 2011, pp. 198-202.

[5] O. E. Omel'chenko, M. Wolfrum, S. Yanchuk, Y. L. Maistrenko, O. Sudakov, Stationary patterns of coherence and incoherence in two-dimensional arrays of non-locally-coupled phase oscillators, *Physical Review E*, Vol. 85, Issue 3, 2012, Paper ID 036210.

[6] O. Sudakov, M. Kononov, Ie. Sliusar, A. Salnikov, User clients for working with medical images in Ukrainian grid infrastructure, *Proceedings of the 7th International IEEE Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications IDAACS'2013*, Berlin September 12-14, 2013, pp. 705-710.

[7] Virtual organization membership service of Ukrainian national grid (UNG), National Taras Shevchenko University of Kyiv, 2011. [Online]. Available: http://grid.org.ua/voms/

[8] Medical Grid-system for population research in the field of cardiology with electrocardiogram database 2012. [Online]. Available http://medgrid.immsp.kiev.ua/

[9] Peter Mildenberger, Marco Eichelberg, and Eric Martin, Introduction to the DICOM standard, *European Radiology*, Vol. 12, Issue 4, 2002, pp. 920-927.

[10] Johan Montagnat, Ákos Frohner, Daniel Jouvenot, Christophe Pera, Peter Kunszt, Birger Koblitz, Nuno Santos, Charles Loomis, Romain Texier, Diane Lingrand, Patrick Guio, Ricardo Brito Da Rocha, Antonio Sobreira de Almeida, Zoltán Farkas, A secure grid medical data manager interfaced to the gLite middleware, *Journal of Grid Computing (JGC)*, Vol. 6, Issue 1, Kluwer, March 2008, pp. 45–59, [Online]. Available http://modalis.i3s.unice.fr/softwares/mdm/

[11] A. Shoshani, A. Sim, J. Gu, Storage resource managers: essential components for the grid, *Grid Resource Management: State of the Art and Future Trends*. Ed. by Jan Weglarz, Jarek Nabrzyski, Jennifer M. Schopf, Boston: Kluwer Academic Publishers, 2003, Vol. 64 of International Series in Operations Research & Management Science, pp. 329–348, ISBN: 978-1402075759.

[12] I. Sliusar, M. Volzheva, Using grid infrastructure as a distributed fault-tolerant data storage for web services, *Proceedings of the Second International Conference on Cluster Computing CC'2013*, Ukraine, Lviv, June 3-5, 2013, pp. 266–268.

[13] R. T. Fielding, R. N. Taylor, Principled design of the modern Web architecture, *ACM Transactions on Internet Technology*, Vol. 2, Issue 2, May 2002, pp. 115–150.

[14] L. Dusseault, HTTP extensions for Web distributed authoring and versioning (WebDAV), Request for Comments: 4918. [Online]. Available http://tools.ietf.org/html/rfc4918

[15] I. A. Sliusar, Automatic replication service for maintaining high availability of data in the grid-infrastructure, *Control Systems and Machines*, No. 4, 2012, pp. 63-69. (in Ukrainian)

[16] B. Koblitz, N. Santos, V. Pose, The AMGA metadata service, *Journal of Grid Computing*, Springer Netherlands, Vol. 6, No. 1, 2008, pp. 61–76.

[17] Libcurl: the multiprotocol file transfer library, 2013 [Online]. Available http://curl.haxx.se/

[18] DCMTK: DICOM Toolkit, 2013 [Online]. Available http://dicom.offis.de/

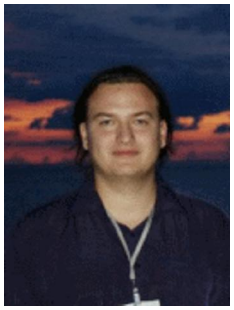
[19] T. Romanenko, V. Vishnevskiy, A. Boretskiy, Features of launching grid task with low computing time and big data transfers in a

distributed repositories of "Medgrid" VO, Proceedings of the International Conference on Parallel and Distributed Computing Systems PDCS'2013, Ukraine, Kharkiv, March 13-14, 2013, pp. 266–268.

[20] M. V. Kononov, O. A. Nagulyak, A. V. Netreba and A. A. Sudakov, Reconstruction in NMR by the method of signal matrix pseudoinversion, *Radioelectronics and Communications Systems*, Vol. 51, Num. 10, October 2008, pp. 531–533.

[21] M. K. Novoselets, S. P. Radchenko, V. A. Tsubin, O. M. Gridko, Ultrasound introscopic image quantitative characteristics for medical diagnostics and refinements of physical noise rise reasons, *Proceedings of the Medical Imaging*, 1994, *SPIE, Physics of Medical Imaging*, Vol. 2163, 1994, pp. 442–446.

Oleksandr O. Sudakov *Candidate of physical and mathematical sciencies (Ph.D.). Associate Professor of Medical Radiophysics Department of Radiophysics Faculty, Head of Parallel Computing Laboratory at Information and Computer Center Taras Shevchenko National University of Kyiv. Graduated from Radiophysics Faculty Taras Shevchenko Kyiv University in 1996. Ph.D. thesis in Procession of magnetic resonance tomography signals in 2002. Scientific interests: high performance computing, physical processes in biological systems.*



Andrii O. Salnikov, *Assistant lecturer at Computer Engineering Department of Radiophysics Faculty, Senior engineer at Parallel Computing Laboratory of Information and Computer Center of Taras Shevchenko National University of Kyiv. Post-graduate of techical sciences.*

Scientific interests: grid and high performance computing.

Ievgen A. Sliusar, *Assistant lecturer at Computer Engineering Department of Radiophysics Faculty, Senior engineer at Parallel Computing Laboratory of Information and Computer Center of Taras Shevchenko National University of Kyiv. Post-graduate of technical sciences.*

Scientific interests: grid and high performance computing.

Oleksandr F. Boretskyi, *Master of physical and mathematical sciences, Postgraduate student of technical sciences, Engineer at Parallel Computing Laboratory of Information and Computer Center of Taras Shevchenko National University of Kyiv. Scientific interests: grid and high performance computing.*

computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

BIG DATA TRANSFER FOR TABLET-CLASS MACHINES

Tevaganthan Veluppillai [1], Brandon Ortiz [2], Robert E. Hiromoto [3]

[1), 3)] University of Idaho, Idaho Falls, Idaho 83402, USA, {hiromoto@uidaho.edu, veluteva@isu.edu}
[2)] University of Idaho, Moscow, Idaho 83844-1010, USA, brandon.ortiz@vandals.uidaho.edu

Abstract: Several well-known data transfer protocols are presented in a comparative study to address the issue of big data transfer for tablet-class machines. The data transfer protocols include standard Java and C++, and block-data transfers protocols that use both the Java New IO (NIO) and the Zerocopy libraries, and a block-data C++ transfer protocol. Several experiments are described and results compared against the standard Java IO and C++ (stream-based file transport protocols). The motivation for this study is the development of a client/server big data file transport protocol for tablet-class client machines that rely on the Java Remote Method Invocation (RMI) package for distributed computing.

Keywords: Blocked file transport; stream-based; block-oriented; tablet machines; client/server configuration.

1. INTRODUCTION

Big data transfer across distributed client/server systems has become a major concern in all fields of information processing. Numerous big data transport utilities have been developed to address the demands in area such as genomics and cancer research [1, 2], high-energy particle physics [3], and GIS analysis [4].

Today's big data file transfer utilities rely on streams of either a single striped file or multiple individual files, whose parallel send/receive client-server strategies are the basis for the increase in big data throughput. The GridFTP File Transfer Protocol [5] is one such. GridFTP is characterized by a fast file transfer protocol that supports large files, secure file transfers, capabilities for multiple destination points for file transfers, and an API that allows various file transfer capabilities. GridFTP is part of the Globus Toolkit.

Current big data transfer tools are designed to make optimal use of hardware parallelism on both the client and server sides of a distributed cluster system. File transport, in such an environment, can be organized into parallel file transfer streams. BBFTP [6], BBCP [7] and the Fast Data Transfer (FDT) [8] utilities are conceptually similar to the GridFTP parallel file streaming approach. FDT, for example, is written in Java and has the capability to run on all major computer platforms. In addition, FDT uses the Java New IO (NIO) [9], where files are processed in blocks-of-bytes rather than the byte-by-byte data stream as performed by standard Java IO.

The introduction of tablet-class machines such as the Apple iPad and Android tablets extends the paradigm of a client-server and opens up the speculation of how big data transfer capabilities can be provided. The challenges posed by tablet-class machines are underscored by their limited hardware resources: the lack of a disk array storage facility, the limited number of communication ports, underclocked processors to reduce heat production, and flash-based memory that are more susceptible to failure. However, one advantage offered by off-the-self tablets is its processor configuration that supports at least a dual-core processing unit. Although limited, this processor parallelism provides a means to implement producer/consumer data transfer strategies. The availability of a producer/consumer data transfer capability and the use of a high-throughput data transfer protocol would provide big data solutions to tablet-class machines. In this paper, we limit the focus to the design of a data transfer utility that can support a tablet-class, client-server environment for big data transfers.

As mentioned above, parallel transfer of large striped-data files provide high throughput by utilities such as GridFTP and FDT. Unfortunately, these approaches require high-end peripheral hardware to capture, coordinate and merge the multiple concurrent streams of a single large data file that arrives at the client-end. Tablet machines are at a

significant disadvantage and as such a more modest yet robust data transfer strategy is required.

This paper is an extension of our presentation made at the IAACS 2013 workshop held in Berlin [10], we limit the focus to the design of data transfer utilities that support a tablet-class, client-server environment for big data transfers. This paper provides a comparative analysis of several Java and C++ approaches that introduces optimizations to reduce the standard Java and C++ IO overheads in filling the socket buffer. Java NIO and Zerocopy [11] are well known techniques described and compared in this paper. A blocked (buffered) Zerocopy is also described and presented as an analog to the NIO method. It should be pointed out that Zerocopy requires a Linux or Unix OS but that should not be an issue on the server side of the network. In addition comparisons with C++ byte-by-byte streaming and a C++ blocked data transfers are used in comparing the different strategies.

In the next sections, assumptions regarding the use of Java New IO (NIO) and Java Zerocopy are described. The proposed approach describes the integration of Zerocopy and NIO with data transfer timing results provided. To complete the comparison, file transfer times between a standard C++ and a block-data transfer are also provided.

2. BACKGROUND

Tablet-class client machines represent a collection of window-oriented technologies for which no unique operating system dominates the industry. In such an environment, the design of a generic client-server data transfer tool must rely on programming languages that are compatible on any and all computer platforms. Java is one such language that bridges this gap but also provides a Remote Method Invocation (RMI) capability that supports coordination between distributed computer platforms seamlessly. The RMI mode can be relatively slow since the instructions are interpreted. However, the use of RMI as a coordinator of a distributed client-server architecture is not a computationally intensive task; for that reason the application of RMI should not incur substantial overhead delays. On the other hand, the internal Java IO stream can lead to TPC/IP overhead delays.

Java is an object-oriented language that employs a byte-by-byte streaming IO process in preparing the socket buffer for data transfer into the network interface card (NIC) buffer and transferred to its destination. At such a fine level of data granularity, Java IO is inefficient and does not scale well.

Block IO data transfers are a more efficient alternative. As such, Java NIO was developed as a block-oriented approach. Java NIO provides the flexibility to adjust the IO block size, which can potentially affect the TPC/IP bandwidth-delay product (BDP) and enhance its throughput performance [12].

In addition to Java NIO, an optimized treatment of internal data copying, known as Zerocopy, was developed and made available in the NIO library. The next two sections will describe the NIO and Zerocopy approaches. The integration of these two methods forms the basis of the desires tablet client-server big data file transfer mechanism.

2.1. JAVA NIO

Java NIO is an open source library developed and maintained by Sun (Oracle). NIO provides block-oriented IO transfers of a file. The strategy of sending a file in a block-wise fashion reduces the software management needs for packetized byte information. The Channel and the Buffer are the principle NIO objects. Channels are analogous to the original Java IO but are bidirectional. In this sense, a channel can be opened for read or for write or for both. Data that is written into a channel must first be written into a buffer, and data that is read from a channel is read into a buffer. A buffer is an object that holds the data that has been read from the channel or holds the data that is to be written into the channel. In the NIO library, all data is handled with buffers. The interplay between the Channel object and the Buffer object marks the operational difference between Java IO and Java NIO. In Java IO, data is written and read directly from Stream objects. NIO allows, therefore, a pipeline between the Channel object and the Buffer object to hide access latencies.

We give two coding examples of NIO that illustrates reading from and writing to a file. A more complete description of NIO can be found in the introductory tutorial [13].

Reading a file requires three steps. First, a channel is acquired by creating a FileInputStream using the original Java IO library:

```
FileInputStream fin = new FileInputStream( "r.txt" );
FileChannel fc = fin.getChannel();
```

Second, a Buffer object is created:

```
ByteBuffer buffer = ByteBuffer.allocate( buff_Size );
```

Third, the data is read from the Channel into the Buffer:

```
fc.read( buffer );
```

This example points out an important aspect of the Channel and Buffer objects. Notice that the

coding of the channel does not explicitly indicate the amount of data that needs to be read into the buffer. As a consequence, Buffer objects are endowed with an internal accounting system that keeps track of the amount of data that has been read and the amount of buffer space remaining for additional data. This capability will be useful in our future implementation of a producer/consumer IO strategy.

The second example is writing to a file. First, a channel is created:

```
FileOutputStream fout = new FileOutputStream( "w.txt" );
FileChannel fc = fout.getChannel();
```

Second, a buffer is created and then data written into it:

```
ByteBuffer buffer = ByteBuffer.allocate(buff_Size );
for (int i=0; i<message.length; ++i) {
    buffer.put( message[i] );
}
buffer.flip();
```

Third, write into the buffer:

```
fc.write( buffer );
```

As in the previous example, the internal accounting system of the buffer automatically tracks the amount of data written into the buffer and the remaining buffer space for which additional data can still be written. The *buffer.put*() method fills the buffer with data, and the *buffer.flip*()method readies the newly filled buffer data to be written to another channel.

Notice that the allocate() method defines the block-oriented IO size. This block size parameter can be dynamically adjusted to affect TCP/IP performance in combination with algorithms such as Fast TCP [14].

NIO supports memory-mapped file IO. This method when applied to reading and writing file data can greatly improve channel-based IO as well as the original Java IO. Memory mapping is an OS capability where the file system maps portions of a file into portions of the memory on demand.

The following code is an example of how a FileChannel() or portions of it can be mapped into memory:

```
MappedByteBuffer mbb =
fc.map(FileChannel.MapMode.READ_WRITE, 0,
buff_Size );
```

The *map*() method returns a *MappedByteBuffer* as a subclass of *ByteBuffer*. Any manipulations using this buffer are automatically mapped to memory on demand by the operating system.

It should be pointed out that the Fast Data Transfer (FDT) IO tools is based in part on NIO.

2.2. ZEROCOPY

Zerocopy is a stream-based file transfer library that differs from Java IO in that it reduces the number of internal data copying and associated context switches. Fig. 1 illustrates the Java IO data copying behavior when a request to send a file from the server to a remote client. The following code represents the data flow encountered in Fig. 1:

```
File.read(fileDesc, buf, len);
Socket.send(socket, buf, len);
```

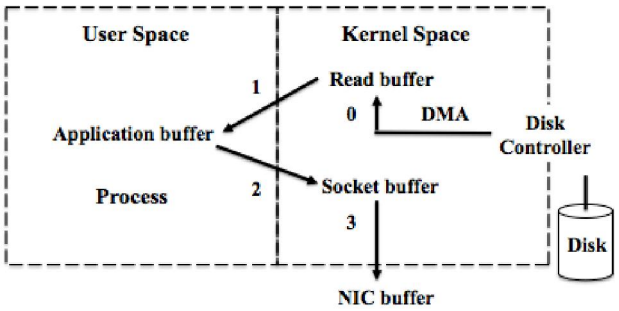


Fig. 1 – Java IO internal data movement & copying [11].

Table 1. Java IO context switching [11].

Time Sequence	User Context	Kernel Context
	Before Read	
T_0		Syscall Read
T_1	Before Send	
T_3		Syscall Write
T_3	Next Cycle	

We see in this illustration that the file's data flow copies the file elements into the Read buffer that is then copied into the application buffer then into the Socket buffer, and finally into the NIC buffer. In order to handle these internal data transfers, the OS intervenes with a corresponding number of context switches. Table 1 lists the temporal order and number of context switches incurred by Java IO. Zerocopy mitigates the number of copying required by Java IO by copying the content of the Read buffer directly into the Socket buffer. The *transferTo*() method in Zerocopy bypasses the Application buffer and copies the Read buffer directly to the Socket buffer. UNIX and various flavors of Linux operating systems support *transferTo*() by routing the method invocation to the *sendfile*() system call. Rather than relying on the two methods File.read() and Socket.send(), Zerocopy is expressed by a single call:

```
transferTo(position, count, writableChannel);
```

Fig. 2 illustrates the Zerocopy approach. From Table 2, the associated number of context switching is reduced from four to two.

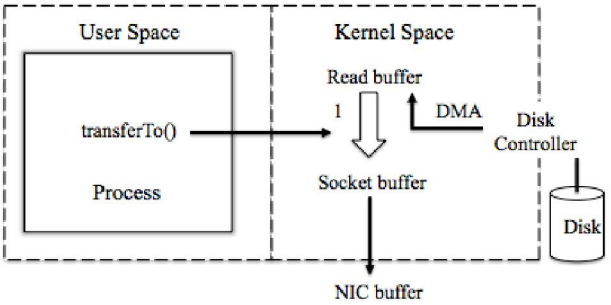


Fig. 2 – Zerocopy internal data movement & copying [11].

Table 2. Zerocopy context switching [11].

Time Sequence	User Context	Kernel Context
	Before transferTo()	
T_0		Syscall Read and Send
T_1	Next Cycle	

2.3. ZEROBUFFER

Unlike NIO, Zerocopy does not rely upon the application buffer to assist in file transfers under the TCP protocol. As a consequence Zerocopy lacks the block-oriented IO structure required to support parallel file transfers, as does NIO. A closer examination of the *transferTo*() method, however, finds that a position index can be set to a beginning location of the file to be transferred. This capability can be used to transform Zerocopy into a block-oriented IO implementation, which is referred to as ZeroBuffer. The following code fragment details the block-oriented ZeroBuffer implementation:

```
fc = new FileInputStream(input).getChannel();

while (position != fc.size())
{
position += fc.transferTo(position, bufferSize, sc);
}
```

The file channel *fc* is initialized and set to the new *FileInputStream*. Within the *while* loop, the *transferTo*() method is invoked with a position indicator that points to a location within the file, a block *bufferSize*, and the socket channel descriptor *sc*.

The complete programs for all the file transfer tests are available at [15].

3. FILE TRANSFER COMPARISIONS

Table 3 provides the detail of the Client/Server machine properties. Table 4 summarizes the file transfer times (milliseconds) between Java IO and Zerocopy for which both use stream-based (byte-by-byte) data transfer methods. The results of these tests indicate a 50 times performance increase of Zerocopy over Java IO. Similar results are reported elsewhere in the literature [11]. The test used file sizes ranging from 0.3 MB to 33 MB. Each file is run ten times to determine a "best" file transfer time. An average transfer time is not reported since the network utilization can distort the significance of average values.

Table 3. Machine functions and properties.

Machine Function	Machine Types	Processor Configuration	Memory System	OS
Server	MacBook	2.8 GHz Intel Core i7	8 GB 1333 MHz DDR3	OS X Lion
Client	MacPro	2 x 2.4 GHz Quad-Core Intel Xeon	16 GB 1066 MHz DDR3 ECC	OS X Lion

NIO and ZeroBuffer are characterized by explicit block (buffer)-size assignments, which from Fig. 1 is referred to as the application buffer. Table 5 lists file transfer times measured for Zerocopy, NIO and ZeroBuffer. Although, Zerocopy is independent of the application buffer size its transfer time is listed for each buffer for comparison purposes only. The best NIO and ZeroBuffer results are listed as a function of their corresponding application buffer sizes, which range from 1 KB to 16 MB. The results presented in this paper are for a 1.03 GB file. Other results for smaller files produced similar results.

The experiments also varied the TCP Send/Recv buffer sizes. The Mac OS X sets the default size of the Send/Recv buffer to 64 KB. The TCP Sendbuffer (on the server-side) is manually adjusted and is coordinated with a reciprocal assignment (on the client-side) for the TCP Receive buffer using the same size.

Table 4. Data transfer times.

Files (MB)	Java IO (ms)	Zerocopy (ms)
File1 (0.3)	7	3
File2 (0.6)	13	7
File3 (1.2)	3687	47
File4 (2.5)	6103	107
File5 (4.9)	14969	298
File6 (9.9)	29307	564

Fig. 3 is a plot of the corresponding file transfer times listed in Table 5. The TCP Send/RecvSpace buffer size is manually set to 64 KB. NIO, Zerocopy

and ZeroBuffer deliver comparable transfer times. There are several items to point out. First notices that ZeroBuffer suffers noticeable overheads for small application buffer sizes (1 KB to 2 KB); whereas, the while-loop introduced to create its block-oriented IO structure shows little difference in comparison with Zerocopy. This is quite surprising given the somewhat "artificial" block-oriented approach taken. The second item to note is the behavior of the NIO method. NIO exhibits file transfer slowdowns for small application buffer sizes (1 KB to 2 KB); as well as for larger buffer sizes (2 MB to 16 MB). Overall, NIO and ZeroBuffer methods are comparable to the file transfer time of Zerocopy, between the application buffer sizes ranging from 4 KB to 1 MB.

Table 5. File size = 1.03 GB (TCP Send/RecvSpace = 64 KB).

Application Buffer Size	NIO (ms)	ZeroBuffer (ms)	Zerocopy (ms)
1KB	109.88	135.15	88.2
2KB	96.62	100.8	88.2
4KB	88.96	88.4	88.2
8KB	89.69	89.05	88.2
16KB	90.29	89.11	88.2
32KB	89.23	89.62	88.2
64KB	90.26	89.12	88.2
128KB	89.79	88.98	88.2
256KB	88.46	88.81	88.2
512KB	88.45	88.36	88.2
1MB	89.97	88.28	88.2
2MB	92.22	88.55	88.2
4MB	94.06	88.45	88.2
8MB	94.55	89.03	88.2
16MB	94.7	88.51	88.2

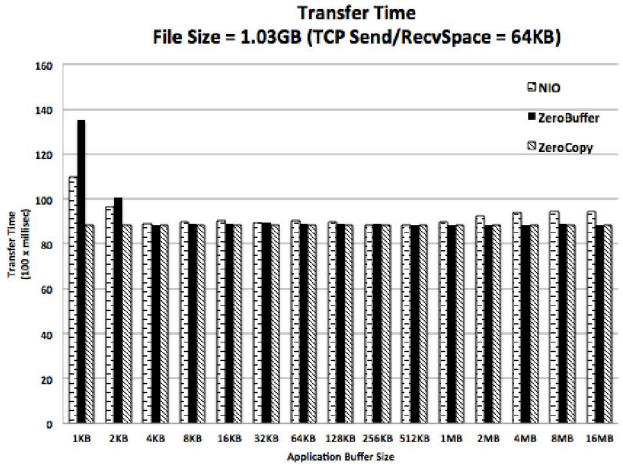


Fig. 3 – A plot of the Table 5.

The TCP Send/RecvSpace buffer settings are considerations based on work by several authors investigating [16, 17, 18] the optimal tuning of TCP file transfers. Starting with the TCP Send/RecvSpace buffer sizes of 64 KB, 156 KB, 256 KB, 512 KB, and 1 MB, we find little difference in file transfer performance. What we observed, however, is that the file transfer performance for NIO improves with increasing TCP Send/RecvSpace buffer size. Fig. 4 and Fig. 5 are presented to illustrate this observation.

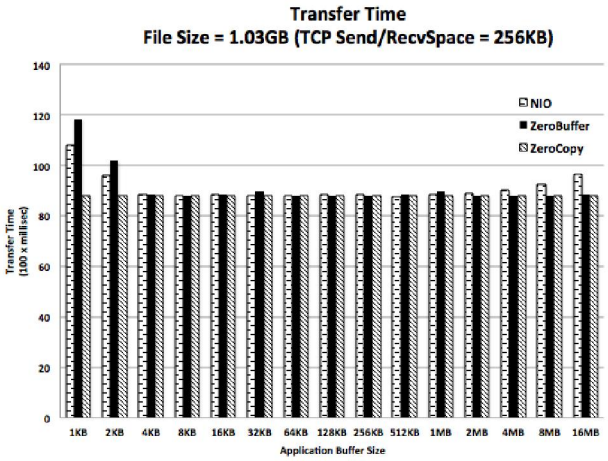


Fig. 4 – Results for TCP Send/RecvSpace = 256KB.

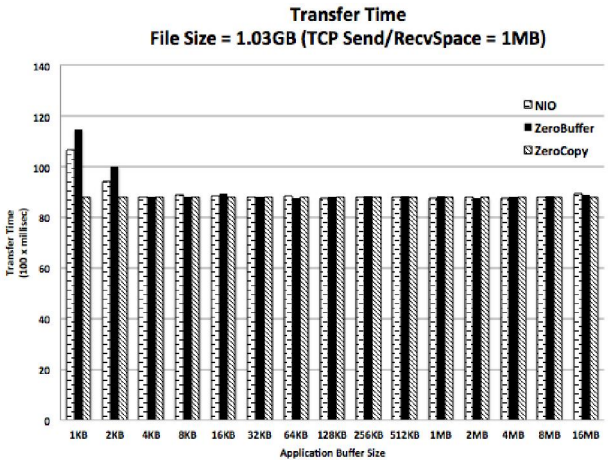


Fig. 5 – Results for TCP Send/RecvSpace = 1MB.

As a final set of comparisons, we examine the transfer speeds that can be obtained with a byte-by-byte and block or buffered data transfer approaches using C++. The two methods are referred to as C++(1) and C++(2), respectively. These methods are analogous to standard Java and Java NIO, whereas a C++ version that is comparable to Zerocopy was not readily available for this study. Fragments of the socket program for each method are provided below [19, 20]. C++(1) uses a simple *while* construct:

```
int ch;
char toSEND[1];

while((ch=getc(file))!=EOF) {
        toSEND[0] = ch;
        send(socket, toSEND, 1, 0);
}
```

C++(2) uses a buffered send size of 1024 bytes as depicted in the following code fragment:

```
char buf[1024];

while (!feof(file)) {
int rval = (int)fread(buf, 1, sizeof(buf), file);
int sent = (int)send(sock, &buf[off], rval - off, 0);
off += sent;
}
```

The TCP Send/RecvSpace for the server processor is maintained at a size of 131,072. On the client side, the TCP Send/RecvSpace is fixed at a size of 65,536. No attempts are made to study the transfer times as a function of the TCP Send/RecvSpace. The sole purpose of this experiment is to compare the relative performance of the various C++ file transfer methods with Zerocopy and ZeroBuffer. The client/server configuration is hosted on a local area network (LAN) within the Center for Advance Energy Studies CAES) in Idaho Falls. We are currently working towards the assessment of file transfer performances over a wide area network (WAN).

Table 6 displays the data transfer times for the different Java and C++ programs. The time is given in milliseconds. The files range in size from 0.256 MB up to 16.4 MB. The buffer size used for ZeroBuffer and C++(2) is set at 1024 bytes.

Table 6. Data transfer times.

File Size (MB)	ZeroBuffer (ms)	Zero (ms)	C++(2) (ms)	C++(1) (ms)
0.256	9	8	1	333
0.512	16	18	2	596
1.0	31	31	4	1158
2.0	55	54	7	2267
4.1	83	85	16	4236
8.2	143	151	34	8273
16.4	161	164	67	16636

Notice that the data transfer rates for the C++(2) implementation is far superior to its byte-by-byte transfer counterpart. This is consistent with results between Java and Java NIO. In either case the block-data transfers utilizes the I/O subsystems more efficiently. This of course is a well-known result. The transfer times of C++(2) varies by a factor of 9 to 2.5 times faster in comparison to the corresponding Zero{Buffer, copy} transfer times; however, notice that the speedup factor decreases with increases in file size. In other words, it appears that for much larger file sizes C++(2) may reach parity with Zero{Buffer, copy} transfer rates. To test this conjecture, we ran two test cases using file sizes of 0.734 GB and 1.47 GB. Table 7 presents the results for C++2, Zero copy and Zero copy + buffer. Again a buffer size of 1024 bytes is used.

Table 7. Data transfer times.

File Size (GB)	ZeroBuffer (ms)	Zero (ms)	C++(2) (ms)
.734	7726	6092	5068
1.47	15194	13332	10098

Comparing the ratios between Zero (copy) and C++(2), the transfer rates of C++(2) now appears to be given by factors of 1.2 (0.734 GB) and 1.32 (1.47 GB), respectively. These results are in good agreement with our prior experiments.

We make one final observation with regards to the transfer of large files. In Figs. 3, 4, and 5 ZeroBuffer consistently underperforms Zero (copy) for buffer sizes less than 4 K bytes. For buffer sizes of 4 K bytes and larger, ZeroBuffer is comparable in file transfer times to Zero (copy). To test the consistency of this behavior, Table 8 list the results of performing the same file transfer experiments but using a buffer size of 5K bytes. For both the 0.734 GB and 1.47 GB files, ZeroBuffer and Zero are comparable in transfer time as argued. What is not expected is the reduction in transfer times for C++(2) by more than 50 %, and between 2.8 to 3.25 speedup over ZeroBuffer.

Table 8. Data transfer times (Buffer = 5 KB).

File Size (GB)	ZeroBuffer (ms)	Zero (ms)	C++(2) (ms)
0.734	6537	(6092)	2010
1.47	12865	(13332)	4583

In Table 8, the times for Zero (copy) are inserted only for comparison purposes (recall that Zero (copy) is buffer size independent).

Determining a more optimal buffer size for C++(2) requires further experiments; although, we suspect that further experiments with larger files and larger buffer sizes will exhibit behavior similar to those illustrated in Figs. 3, 4, and 5.

4. SUMMARY AND CONCLUSION

We developed data transfer programs using commonly available socket libraries. The socket programs for NIO, Zerocopy, and C++ are straightforward and required no special programming considerations. The timer resolution of the C++ programs is based on the Apple LLVM compiler and developed under the XCode Integrated Development Environment (IDE) [21].

The comparative collections of I/O results are presented. The programs build upon standard Java and C++ techniques and libraries available to the general users. More specialized routines might be found at advanced application websites or through proprietary sources.

The data streaming strategy of Java IO and C++ are known to be inefficient for big data transfers. In this study, block-oriented IO methods are compared to an efficient non-blocked IO method known as Zerocopy. In addition, this paper provides the first direct comparison between Zerocopy and NIO.

A block-oriented IO method (ZeroBuffer) is introduced that supports Zerocopy efficiencies over a large range of application and TCP buffer sizes. Although the Zerocopy method can be reliably used to address the appetite of tablet-class client/server file transfers, ZeroBuffer has the potential to support non-blocking, concurrent IO threads, which is an important feature of high-end grid-IO.

Overall, the buffered C++ implementation proved to be the fastest of the file transfer codes, but required a buffer size greater than 4 K bytes. ZeroBuffer is shown to have similar behavior; although, not as fast.

A further advantage may be gained when clients and server are hosted on a WAN where the bandwidth utilization for the transfer of large data files is strongly influenced by the behavior of the TCP network protocol under extreme external demands. In this regard, the block-data transfer mode can provide a self-throttling mechanism to reduce the TCP overhead latencies experienced by large data transfers initiated in a single-continuous send-operation mode over the network.

Finally, this paper represents the initial stages of a much more ambitious effort to provide serious information discovery and analysis on tablet-class machines. To achieve this goal, a new paradigm of big data transfer should be considered and realized: raw data should never be transferred across wide area networks. Instead raw data should be localized in persistent data servers where stored data should only be marshaled into a form that is information-dense; that is, a form that can be easily visualized, virtualized, and orders of magnitude smaller in size than its original raw data footprint. Optimal data transfer time or data compression is not enough to achieve this goal. A big data system will likely require block-algorithms to sustain a producer/consumer-like pipelined data transfer protocol, where blocks of data are computed and transferred across the network in an overlapping pipeline fashion. In order to sustain this operation, a distributed server farm that coordinates the in-situ placement of raw data and its access will prove to be advantageous.

5. ACKNOWLEGMENTS

This research is being performed using funding received from the DOE Office of Nuclear Energy's Nuclear Energy University Programs.

6. REFERENCES

[1] C. Wang, D. Zhang. A novel compression tool for efficient storage of genome resequencing data, *Nucleic Acids Research*, Vol. 39, Issue 7, April 2011, e45.

[2] Lynda Chin, William C. Hahn, Gad Getz, et al. Making sense of cancer genomic data, *Genes & Development*, Vol. 25, Issue 6, 2011, pp. 534-555.

[3] http://supercomputing.caltech.edu (last accessed 15, Feb., 2013.)

[4] M. J. de Smith, M. F. Goodchild, P. A. Longley. *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools*, 2nd edition, Troubador, UK, 2007.

[5] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The globus striped GridFTP framework and server, *Proceedings of the ACM/IEEE conference on Supercomputing SC'05*, ACM Press, November 2005, pp. 54.

[6] http://doc.in2p3.fr/bbftp/

[7] http://www.slac.stanford.edu/~abh/bbcp/

[8] R. S. Prasad, M. Jain and C. Dovrolis, Socket Buffer Auto-Sizing for High-Performance Data Transfers, *Journal of Grid Computing*, Vol. 1, Issue 1, 2003, pp. 361-376.

[9] Getting started with NIO, https://www.ibm.com/developerworks/java/tutorials/j-nio/j-nio-pdf.pdf.

[10] V. Tevaganthan, B. Ortiz, R. E. Hiromoto, A big data file transfer tool for tablet-class machines, *Proceedings of the IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2013)*, Berlin, Germany, September 12-14, 2013, pp. 676-680.

[11] S. K. Palaniappan and P. B. Nagaraja, Efficient data transfer through zero copy: Zero copy, zero overhead, 2 Sep 2008, http://www.ibm.com/developerworks/library/j-zerocopy/

[12] M. Jain, R. S. Prasad and C. Dovrolis, The TCP bandwidth-delay product revisited: network buffering, cross traffic, and socket buffer auto-sizing, http://hdl.handle.net/1853/5920.

[13] Greg Travis, Getting started with new I/O (NIO) skill level: introductory,

http://www.ibm.com/developerworks/java/tutorials/j-nio/j-nio-pdf.pdf.
[14] C. Jin et al., FAST TCP: from theory to experiments, *IEEE Network*, Vol. 19, Issue 1, 2005, pp. 4-11.
[15] http://datatransfer.codeplex.com
[16] S. Thulasidasan, W. Feng, M. K. Gardner. Optimizing GridFTP through dynamic right-sizing, *Proceedings of IEEE International Symposium on High Performance Distributed Computing*, 2003, pp. 14-23.
[17] J. Semke, J. Mahdavi, M. Mathis, Automatic TCP buffer tuning, *Computer Communication Review*, Vol. 28, 1998, pp. 315-322.
[18] R. S. Prasad, M. Jain, C. Dovrolis, Socket buffer auto-sizing for high-performance data transfers, *Journal of Grid Computing*, Vol. 1, Issue 4, 2003, pp. 361-376.
[19] *Beej's Guide to Network Programming*, Jorgensen Publishing, October 20, 2011.
[20] http://www.linuxhowtos.org/C_C++/socket.htm (last accessed 16 Dec., 2013).
[21] https://developer.apple.com/technologies/tools/ (last accessed 31 Dec., 2013).

Tevaganthan Veluppillai *is a Master's student at the University of Idaho. He is developing of client/server system for tablet-class client machines.*

Brandon Ortiz *is a PhD student at the University of Idaho. His interests are in the development of wireless communication protocols for autonomous vehicles.*

Robert E. Hiromoto*, received his Ph.D. degree in Physics from University of Texas at Dallas. He is professor of computer science at the University of Idaho. His areas of research include information-based design of computational algorithms, and information processing applied to decryption techniques and secure wireless communication protocols.*

computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

VECTOR CLOCK TRACING AND MODEL BASED PARTITIONING FOR DISTRIBUTED EMBEDDED SYSTEMS

Robert Hoettger, Burkhard Igel, Erik Kamsties

University of Applied Sciences and Arts
Pimes Research Institute
Fachhochschule Dortmund,
Sonnenstr. 96, 44139 Dortmund, Germany
robert.hoettger@fh-dortmund.de, igel@fh-dortmund.de, erik.kamsties@fh-dortmund.de

Abstract: Tracking, partitioning and tracing in modern dynamic high performance computing systems are three of the most innovative and important development aspects for performance optimization purposes and state-of-the-art advanced quality. This paper discusses these three aspects with respect to distributed systems and proposes new mechanisms for an advanced utilization of software in this domain.

We present a specific tracking mechanism via vector clocks for model and code partitioning purposes and the determination of causality relations. Further, a tracing approach for an effective analysis and thereby utilization of code and the corresponding architecture is introduced. The combination of both approaches leads to a high degree of parallelism and a fine-grained structure of execution units, that further traced, supports a precise analysis of synchronous and asynchronous system's behavior as well as an optimal load balancing. The mechanisms are introduced with respect to a model based control engineering tool and event diagrams.

Keywords: partitioning; event tracing; vector clocks; control engineering; distributed systems; virtual time.

1. INTRODUCTION

The modern digital computing era involves increasing amounts and relations of stored data as well as more complex computation platforms, architectures, tools and frameworks. A common, mandatory and important aspect is the prevention of computation- and storage overheads i.e. the efficient use of soft- and hardware. Especially the modern distributed system domain reveals a more complex determination of causality relations due to the use of replicas, unreliable hardware, large scales of data and commoditized machines. The increasing number of requirements, functions, safety issues or assistance demands call for a significant increase of computing power accompanied by the request for reduction of energy and costs. To handle these requirements the multicore processor technology starts to permeate electronic control units (ECUs) in cars for example. Existing applications cannot realize immediate benefit from these multicore ECUs, because they are not designed to run on such architectures.

The Itea 2 project 09013 *AMALTHEA*[1] is a state of the art research project in the automotive industry that addresses building a model-driven platform for this new generation of development environments, which supports the development of multicore systems, takes product line engineering into account and produces AUTOSAR [1] compatible software. Tracing and partitioning are two of the challenges to be met with respect to timing constraints. This paper presents a novel approach for both partitioning and tracing and further supports the determination of causality relations among events in a distributed system. Multicore systems in this case are one example for distributed systems.

Partitioning in context directed acyclic graphs that occur in most computing applications, influence system performance. The more efficient the partitioning process forms computation sets distributed among computation units i.e. processors, the more the systems benefits from time issues, energy demands or high performance real time applications. These aspects are common topics of interest in almost all areas of science and technology.

Forming computation sets mostly concerns the division of processes into subprocesses whereas each subprocess consists of computational load [2]. In terms of graph theory these subprocesses are

[1] Itea 2 09013 AMALTHEA, BMBF funded.

denoted as nodes. A node often reveals uni directed communication with one or multiple other nodes, such that a directed transition between them denotes dependency as shown in Fig. 1.

Fig. 1 – Node dependency.

Node B depends on a result of Node A and thereby depends on Node A. In case Node B is assigned to a different computation unit i.e. processor, the system must preserve the given ordering of both nodes. Otherwise Node B may be started without Node A being finished resulting in Node B termination violation. Such order may be preserved via inter process activation, client / server calls, OS-events (Node A (set) and Node B (wait)) or Semaphores.

The paper is organized as follows. The next section introduces related work on tracking, partitioning and tracing. Afterwards the concrete usage of vector clocks is described with respect to a model driven control engineering example, that is adapted to several different partitions and load balance approaches. The following section introduces tracing as a more comprehensive approach for performance and partitioning optimization purposes. Finally, the novel approach is analyzed according to benefits, ease of use and industrial relevance. Corresponding contents are published with respect to [3].

2. RELATED WORK

Innovation according to *virtual time*, *tracing* and *partitioning* stretches over years of development and huge amounts of different mechanisms and algorithms addressing the increasing number of requirements and constraints emerging from all kinds of political, entertainment, safety or energy demands.

Tracking, initially used in technical and theoretical computing, considers causality relations and the determination of event orderings in distributed systems with the help of logical clocks [4]. Extending this mechanism in order to gain information about the program's global state and possible concurrency, vector clocks can be used [5, 6]. Newer approaches focus on applying the exposed mechanisms to models, transferring models to mathematical equations [7] or introduce graphical editors, model checkers, code generators, simulators or dynamic systems and algorithms [8]. All these mechanisms basically address the derivation of timing characteristics for causality relations and thereby ensure logical and temporal correctness within communication, synchronization and computational flows actively by applying the certain mechanisms to a system.

Partitioning is a significant approach for an efficient assignment of runnables to tasks in order to utilize parallel computing. The generic PCAM (Partitioning, Communication, Agglomeration and Mapping) approach by Foster et al. [9] forms the basis for most common partition approaches. It focuses on providing benefits like improving cost-performance ratio, availability via avoiding redundancy, computing power and understanding of a program's behavior due to more detailed information about the problem structure. Partitioning is a division of independent parts in order to solve them in parallel. Therefore, small tasks must be defined, that utilize processors in an optimal way and avoid duplicate data and calculation. The smaller the partitions get, the more flexible and potential the parallelism is. Foster [9] further introduces domain decomposition and functional decomposition. In domain decomposition, data associated with a problem is divided into small parts with approximately equal size. Afterwards, computation is partitioned by associating the operations with the data on which it operates. The focus within functional decomposition lies on the computation that is to be performed instead of the data that is manipulated by the computation. The computation is divided into disjoint tasks, with a subsequent data requirement analysis. In case the data requirements are disjoint, the partition is complete, otherwise considerable communication is required to avoid data replication.

Tracing addresses revealing a program's execution according to more complex problems, errors, ineffective patterns and a lot more issues by considering way more parameters like architecture properties, scheduling paradigms, signals, runnables, processes, or threads and corresponding timing properties depending on the used trace format. Though, tracing only passively applies optimization and efficiency on a system, as specific trace format's APIs are used to generate trace files that can be read by specific tools. These tools mostly reveal the system's behavior in a timeline diagram and users are supposed to react and improve systems according to conflicts and ineffective patterns.

3. PARTITIONING

The following sections propose a novel approach for distributing execution units, emerging from both model elements i.e. data flow systems and vector clock traces as a result of specific code extensions.

The mechanism is based upon a transformation to a data flow graph (*directed acyclic graph*) and a subsequent partitioning for either a dynamic number or a fixed number of processes. The final result leads to an optimal utilization of parallel resources.

3.1. DATA FLOW PARALLELISM

Typical data flow systems provide a fine grained early degree of parallelism. Having a data flow system like Fig. 2, delay blocks encapsulate calculation dependencies providing distributed calculations due to their output being not directly dependent of their input.

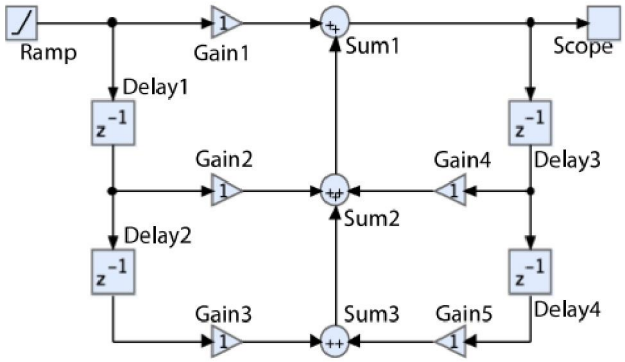


Fig. 2 – Data flow diagram.

The data flow diagram shown in Fig. 2 is derived via the frequency response:

$$H(z) = \frac{Gain1 + Gain2 \cdot z^{-1} + Gain3 \cdot z^{-2}}{1 + Gain4 \cdot z^{-1} + Gain5 \cdot z^{-2}} \quad (1)$$

In the first step i.e. calculation cycle, *Ramp* and *Delay1* to *Delay4* can be calculated in parallel by either different runnables, tasks or cores. The second step contains all blocks connected to the encapsulated blocks of the first step i.e. *Gain1* to *Gain5*. The fact that the subsequent components hold more than one input i.e. dependencies of previous components, *Sum3*, *Sum2*, *Sum1* and *Scope* must be executed subsequently after the first two calculation cycles (see *data flow graph* Fig. 3). Such calculation cycles may be also known as sequential code segments (SCS) [10].

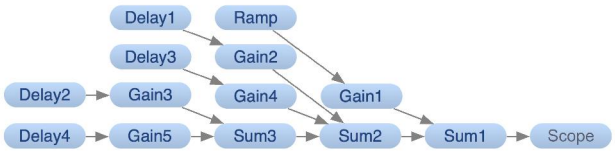


Fig. 3 – Typical DFG with nodes (blocks) and transitions.

3.2. DATA FLOW GRAPH

A data flow graph (DFG) displays nodes (execution units, algorithms, calculations, functions, events, blocks, etc.) connected to other nodes via transitions or edges sequentially (often from left to the right) and is convenient for the exploration of parallelism due to its asynchronous nature. It can be both created via modeled block diagrams (control engineering) or via vector clocks augmented code (described in Section III-C) and can be transformed to a table based structure revealing mandatory sequential orderings, dependencies (horizontally, indicated by arrows) and concurrency (vertically). Any DFG usually exposes a directed acyclic graph structure such that DFG(N,T,R,S) is defined by N nodes (execution units), T transitions, R root nodes and S sink nodes with no directed cycles. Any node $n \varepsilon N$ lies at least within one path from a root node $r \varepsilon R$ to a sink node $s \varepsilon S$. Any transition $t \varepsilon T$ between two nodes n_1 and n_2 represents data dependency between the two nodes and implies mandatory sequential ordering such that the execution of n_1 precedes n_2 in time. Hence n_1 and n_2 shall not be mapped to different processes or processing units as they can not be calculated in parallel.

A node n may possess multiple in- and out-transitions and transitions must always cross one or more sequential code segments (SCS). A low level DFG can be seen in Fig. 3 exposing the data flow model of Fig. 2, featuring R = (Delay1, Delay2, Delay3, Delay4, Ramp), S = (Scope), $\#T = 13$ and $\#N = 14$. Such DFGs can be automatically created from any block diagrams such as in the *Damos* environment [11]. A critical path leading from a root r to a sink s provides the maximal number of sequential nodes and represents the minimal runtime of a program. There may exist several critical paths in a DFG. One possible critical path in Fig. 3 starts at *Delay4* and ends at *Scope* (the other critical path in this example starts at *Delay2* and ends with *Scope*). Any usual control engineering based blockdiagram can be transformed into a DFG via forming SCSs with the help of delay calculation encapsulation and depth first search calculations. A DFG is mandatory for an optimal partitioning, respectively efficient utilization of distributed resources as described in Section III-D. The process of finding the critical path starts with identifying the root and checking all dependent nodes, whether one or more nodes provide the distance of the root 1 to the farthest sink. Afterwards for all selected nodes that provide that distance, the process is repeated regarding the selected node's sink distance 1 until the sink is found. This methodology identifies at least one

critical path via a helping function, that calculates a node's distance to the farthest sink.

3.3. VECTOR CLOCK AUGMENTED EXECUTABLE CODE

Any program code can be partitioned to one or more processes or initially execution units featuring dependencies. Most common programs use message passing techniques and data transfer between functions, objects and similar execution units for communication. In case such a program is not related to a modeled system (like described in Section 3.1), one can extend any program's code, adding vector clock API calls at specific points for both creating vector clock traces and use validation mechanisms for tracking data updates and determine causal dependency relations among transactions. Data updates thereby support synchronizing events in a totally decentralized way. Especially modern transactional systems using partial replication and scalable distributed multiversioning such as NoSQL data grids like BigTable, Amazon Dynamo or Cassandra require multiversion based update mechanisms [12]. A simple vector clock trace in combination with the executable code can be used in order to create an unpartitioned DFG on the one hand as well as a partitioned message passing based event diagram on the other hand. Such an event diagram is shown in Fig. 4 as an example, featuring three processes and several communicating events.

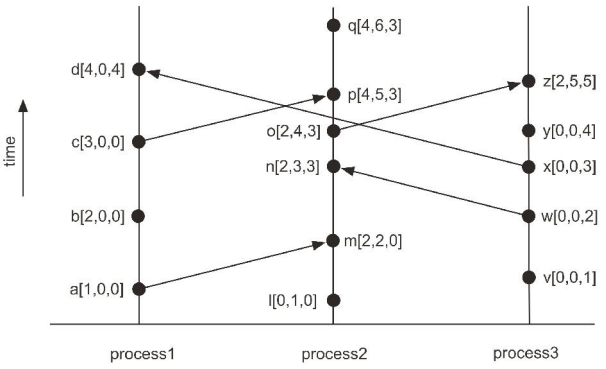


Fig. 4 – Typical event diagram with three processes.

It is assumed, that processes communicate through message passing in a classic asynchronous way such that messages consume a specific delay. The three processes show parallelism, whereas the determination of causality relations (provided by the vector clocks) respectively the knowledge of the precise time related occurrence of events and their communication is mandatory to avoid conflicts and preserve the program's semantics. The mentioned mechanism for causality relation determination of events was introduced by [5] and [6] simultaneously via comparing vector clocks using the following rules:

$$e_1 \rightarrow e_2 \; means \; C_{e1}[p] < C_{e2}[q] \tag{2}$$

and vice versa:

$$C_{e1}[p] < C_{e2}[q] \; means \; e_1 \rightarrow e_2 \tag{3}$$

Here, e_1 and e_2 define two different events with corresponding vector clock arrays C_{e1} and C_{e2}, $\rightarrow$ defines the "happened before" relation and p, q define two transactional processes.

A generated vector clock trace already references a specific number of processes and can be used in order to assign the code segments to different processes i.e. to perform the partitioning. A vector clock trace can be extended as described in Section 4. Without a vector clock trace, the described DFG is a central activity for partitioning and load balancing. A DFG can be created via identifying nodes (execution units) and transitions (dependencies).

In order to gain a partitioned event diagram (see Fig. 4) from an unpartitioned DFG (see Fig. 3), all execution units need to be assigned to processes. This can be dynamically performed assuming a static predefined number of processes. Each API call then assigns the execution units chronologically to a process with respect to their communication. Assuming execution unit a with transactions to *b* and *m* (Fig. 4), *m* could be assigned to process2 or process3. Process2 is chosen if execution unit *l* at process2 finished. If *l* is not finished, process3 is chosen if execution unit *v* finished. If both processes are performing calculations (execution units *l* and *v*) at execution unit *m* assignment time, the event will be assigned to the process, that notifies its availability first. The vector clock mechanism thereby ensures the correct replication of the actual behavior i.e. the call sequence in a distributed system. This mechanism is important especially for distributed systems consisting of multiple commoditized systems, meeting the necessity of a global time for causal ordering determination e.g. managing consistency in the Amazon Dynamo architecture [13]. The actual DFG for the event diagram in Fig. 4 is shown in Fig. 5. The DFG reveals an optimal parallelism of three processes, providing a complete system calculation consisting of 15 nodes in six SCSs (steps) and five cross process communications indicated by dashed arrows.

The proposed executable code trace generation extension provides causality relation determination as well as the DFG- and event diagram partitioning

approaches for an efficient parallelism support via trace analysis.

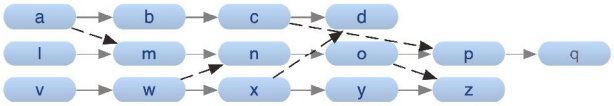


Fig. 5 – Event diagram correspondent DFG.

3.4. DFG PARTITIONING

Besides the use of the vector clock API call augmented executable code for program partitioning and distribution, data flow systems can utilize similar mechanisms in order take advantage of parallelism. Data flow diagrams (Fig. 2) can be transformed to data flow graphs (DFG, Fig. 3) as described in Section 3.2 in order to apply a specific partitioning mechanism for assigning nodes to runnables or processes. The DFG's number of SCSs defines the minimal number of steps (sequential executions) and the number of rows defines the maximal number of processes, whereas the number of occupied rows varies from SCS to SCS and the maximal process number refers to the SCS with the maximal row count. Fig. 6 displays the event diagram with regard to the data flow example shown in Section 3.1 i.e. Fig. 2 and Fig. 3 partitioned to four processes. Here, the maximum number of sequential nodes is bound to process1 by six nodes.

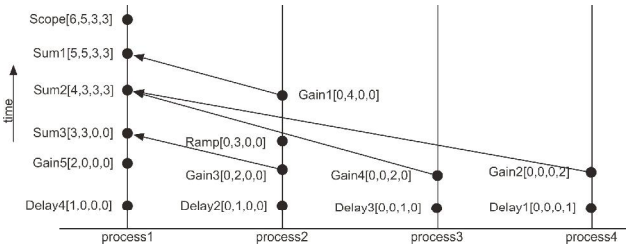


Fig. 6 – Data flow correspondent event diagram with four processes.

Fig. 7 displays the same program, but mapped into three processes. Here, the maximum number of SCS is increased to seven due to the fact that process3 can not calculate *Delay1* at the third SCS because of *Delay1*'s adjacent nodes to the sink. The partitioning algorithm always assigns nodes to a process according to the SCS and the node's adjacent nodes to the sink. Assigning nodes to process3, the algorithm only detects *Gain2* for SCS three (fourth last SCS to sink), due to *Delay1* (being the only unassigned node besides *Gain2*) revealing four adjacent nodes to sink and only nodes with maximal three adjacent nodes to the sink are considered. In this case the partitioning algorithm stretches processes in order to assign the unassigned nodes, i.e. inserting a new SCS at the corresponding SCS step beginning with with first process that is a new SCS preliminary to Sum3 in the example shown in 7.

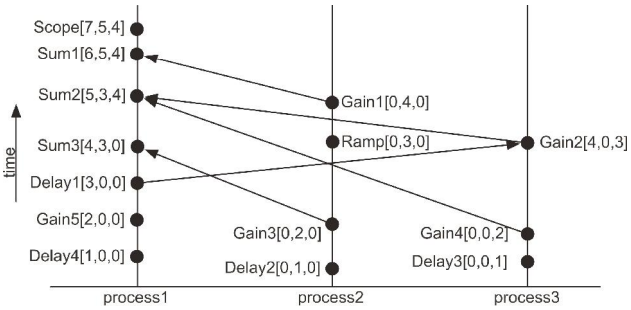


Fig. 7 – Data flow correspondent event diagram with three processes.

Having the nodes assigned to two processes, the result looks like Fig. 8. Here, the maximum number of SCS rises to eight due to two stretch operations caused by *Delay1* and *Gain2*.

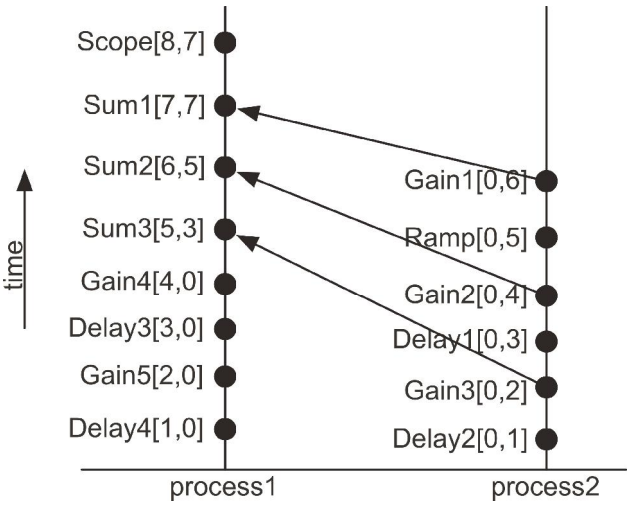


Fig. 8 – Data flow correspondent event diagram with two processes.

The example shows, that a critical path described in 3.2 with exactly one node from each SCS connected via transitions initially forms the first process. The partitioning algorithm is supposed to determine the critical path, that provides the least cost instensive calculation, for forming the first process. Hence it uses multiple optimization criteria i.e. minimal runtime, minimal cross process communication and the precise parallelism degree (number of processes constraint). In order to form additional processes, already assigned events are ignored and the farthest node from a sink is identified and assigned to another process via a depth first search (DFS). According to the following node assignments to each SCS at a process, firstly adjacent nodes (that provide a transition to the

preceding assigned node) and secondly any other nodes according to the specific SCS are considered. In case neither a adjacent nor a node for the specific SCS can be found, nodes from subsequent SCSs are taken into account. In case the number of processes is restricted by the user and the mechanism obtains unassigned nodes, the partitioning algorithm is able to stretch processes and insert unassigned nodes at specific SCS steps in order to finally assign all nodes. These assignment are processed with regard to the node's dependencies and execution cycles, such that no order constraint is violated.

The previously described methodology has been implemented in an approach called local graph partitioning (LGP). The basic idea of LGP is the assumption of at least one critical path within a directed acyclic graph. This path represents a sequential ordering that does not benefit from being distributed or calculated in parallel due to each node depending on previously calculated results. Mapping such a critical path to different calculation units would result in an increased calculation time due to overheads produced by synchronization and communication between the calculation units. Consequently the critical path is assigned to the first ProcessPrototype and all side paths, branches, sources and sinks of the graph are calculated parallel to that critical path in other ProcessPrototypes. The amount of ProcessPrototypes, respectively the number of actual parallel calculations, can either be maximized automatically by the implementation or specifically defined by the user. The partitioning is able to identify the maximal number of nodes to be calculated in parallel and creates ProcessPrototypes correspondingly. Furthermore, in case the user defines a specific number of threads, the partitioning is able to stretch threads by inserting nodes at a specific time slice between already assigned nodes according to their distances to the farthest sink in order to meet the user's thread constraint. The LGP mechanism shall be outlined by the following pseudo code.

Initially, in lines 1 and 2, two sets are built, containing unassigned nodes and all tasks. Afterwards the critical path is determined, assigned to the first task and all critical path nodes are removed from the list containing the unassigned nodes (*U*). The subsequent for loop (line 4) performs the node to ProcessPrototype (task) assignment such that other ProcessPrototypes contain graph branches beginning with the runnable (node) that provides the greatest distance to the critical path's sink. In case the number of ProcessPrototypes has been automatically calculated, this assignment will cover all occurring runnables. The second for loop (line 8) assigns a node from the list that contains the unassigned nodes (*U*) to each time slice parallel to the critical path with respect to not violating any order constraint. The subsequent while loop (line 24) performs the node insertion process, that is activated in case the user restricted the number of tasks to a smaller value compared with the automatically generated value. In other words, the loop will only be executed in case there remain unassigned nodes after the prior node to task assignment. The user's task number restriction causes each task to execute more nodes such that the overall execution time will be greater than the critical path's execution time. This *stretching* (execution time increase) is defined by the *stepincrease* value (see Listing 1 lines 25-28). The *stepincrease* value is calculated by the number of unassigned nodes divided by the number of tasks and incremented in case the division did not result in an integer value. This ensures that all unassigned nodes can be evenly distributed among the tasks. E.g. if there are three tasks and five unassigned nodes, the tasks one and two will be extended by two nodes (5=3 = 1+1 = 2) and task three by one node.

```
1  Let U denote the set of all unassigned nodes
2  Let T denote the set of tasks
3  Determine the critical path CP and its length CPL (number of
       nodes) and remove the CP nodes from U
4  FOR each task t in T
5    Determine the farthest node to the next sink fn in U
6    Let NTSi denote fn's distance to the farest sink (critical
         path sink)
7    Let NTSo denote fn's distance to the next source (critical
         path source)
8    FOR each time slice s of CP in t beginning with s=CPL
9      IF NTSi == s
10       select fn
11     ELSE IF there are multiple farthest nodes
12       Let fns denote all nodes in U providing the farthest
             distance to sink
13       select fn of fns with highest NTSo
14     ELSE IF there is exactly one farthest node && NTSo <=
           CPL-s+1
15       select fn
16     ELSE no node fits into current time slice --> empty slot
17     ENDIF
18     Assign selected node to t
19     remove selected node from U
20     set fn to either a connected node providing a distance to
           sink = fnd-1 or to the node providing the farthest
           sink distance
21   ENDFOR
22 ENDFOR

24 WHILE U is not empty
25   set stepIncrease to U.size / T.size
26   IF U.size modulo T.size not equals 0
27     increase stepIncrease by one
28   ENDIF
29   FOR each stepIncrease
30     FOR each task t
31       IF U is not empty
32         Determine the farthest node to the next sink fn in U
33         Let fnd denote the distance to the next sink of fn
34         assign fn to t after fnd
35         remove fn from U
36       ENDIF
37     ENDFOR
38   ENDFOR
39 ENDWHILE
```

Listing 1. Pseudocode for partitioning algorithm

A feature that is not mentioned in the pseudo code is the recognition of CPC (Cross Process Communication). For instance if a runnable A provides a RunnablePrecedence to a runnable *B* and the runnables are assigned to different ProcessPrototypes, specific model elements have to be created as described in the introduction (section I) i.e. synchronization events and sequencing constraints.

Finally the LGP aprroach provides all system's runnables distributed among several ProcessPrototypes (user defined or automatically generated) as well as explicit CPC model elements, that can be combined and transmitted to a mapping plugin [14] for further information augmentation and finally to a code generator in order to apply the partitioning to the software and run it in parallel on a multicore system. The partitioning mechanism processes runnables (nodes) with respect to their dependencies (orderings) and execution cycles and utilizes multicore architectures by efficient parallelism and load balancing such that execution times and energy consumption can be lowered and high performance application development can be facilitated.

3.5. PARTITIONING EVALUATION

In [10] two several similar approaches to DFG partitioning are introduced with respect to node's earliest and latest initial times respectively node's runtime or calculation cycles. However the particular number of processes constraint is not considered i.e. merging untreated nodes into existing processes. Due to strictly forming and not changing the critical path, the partitioning in [10] is ineffective according to calculation intense multiple propagated nodes in combination with a process amount constraint. Hence the proposed DFG partitioning in this paper benefits from parallel constraint consideration. Fig. 9 shows *a)* a DFG and correspondingly in *b)* a pipeline partitioning, in *c)* the partitioning from [10] and in *d)* the proposed partitioning of this paper for two processes (indicated by the lower row as process1 and the upper row as process2). The dashed arrows indicate process wide communication. The pipeline partitioning features most cross process communication due to not considering any dependencies. The c) partitioning features the critical path in process1 but increases the overall SCSs due to not being capable of stretching a process. The presented partitioning approach of this paper is shown in d) and provides both a low SCSs amount as well as low cross process communication. The presented partitioning reveals a simple structure whereas industrial applications feature much bigger DFGs such that the partitioning provides more significant benefits for parallelism.

Several literature emphasizes on reducing communication overhead like the region partitioning approach [15] or min-cut partitions [16] or distinguishing between control-, data- and dependence transitions [17] via specific complex mechanisms whereas the presented approach of this paper focuses on simplicity and an efficient automatic load balancing for practical issues in early development phases.

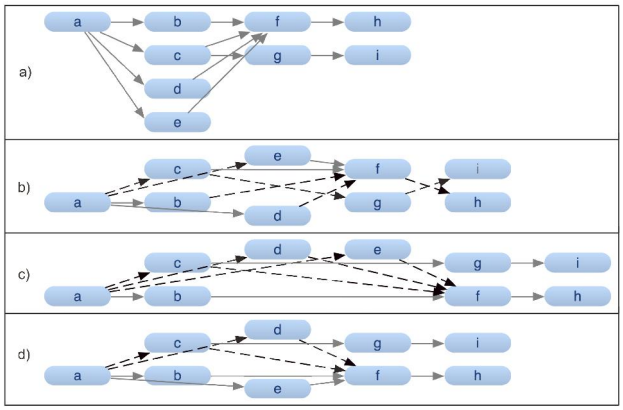


Fig. 9 – Comparison of DFG partitioning.

4. PERFORMANCE OPTIMIZATION THROUGH TRACING

The next step after the efficient and causal correct partitioning, is addressing and revealing more convoluted problems like race conditions, errors, ineffective patterns and dynamic behaviors by considering system parameters like architecture properties, scheduling paradigms, signals, runnables, processes or threads and corresponding timing properties. This can be handled by storing relevant system activities during execution of a physical or simulated system by using a extended trace API, that specifies the trace format. An additional program is supposed to read and analyze the trace data, that has been recorded during the system's execution.

The *AMALTHEA* project takes advantage of two major types of evaluation applied to the trace data, the metric calculation and the Gantt visualization. The metric calculation determines response times of a task or the duration of event chains for instance. All these metrics have in common that certain actions of specific entities are required for calculation. The response time of a task for example requires the actions activated and terminated from the related task. Another example is an event chain which consists of a write access of a task *A* and a read access of task *B*. This case requires the collection of both tasks' related events.

The evaluation types require both dynamic system behavior exploration and dynamic system behavior comparison. The dynamic system behavior exploration allows the determination of system characteristics during execution of the system by tracing system environment or system parts interaction. It provides system behavior, resource consumption, safety related activities and the generation of the system's model. The dynamic system behavior comparison provides quantifying

differences between modeled and physical systems at different development phases (architectural design, functional design, implementation, verification) in order to improve the abstract modeled system.

A vector clock extended trace intensely improves handling simulation, inferential performance and error analysis due to abstraction via virtual time and no need of global clocks respectively the timestamp. Furthermore, tracing execution time for software elements facilitates the partitioning activity by detecting relevant execution time differences for the same SCS. Knowledge about execution times for all nodes improves the load balancing via assigning appropriate nodes to empty time slots at different SCSs (before a cross process synchronization or data exchange for instance). Besides vector clocks the new trace approach features various information like the timestamp, its resolution, a configuration section for comments, optional parameters, creator and format version, the event type, a trace merge field, a memory access field, a memory protection usage field, the recording's precision, the unique event identifier, an instance field and provides pre-defining the data set to be logged.

Having all this information, one can gain absolute knowledge about a system's execution respectively use key information in order to evaluate, improve, and optimize a system. This especially concerns performance analysis according to preserving the temporal and spacial relationships of events, gaining information about using limited resources more efficiently or increasing scalability for bigger simulations. Vector clocks in this context facilitate causality relation determination and constitute a way of replacing expensive timer modules as well as combined with the described trace data, provide the detection of inadequate states during runtime in contrast to debugging, that stops the system at specific breakpoints.

5. CONCLUSION

The proposed new partitioning mechanism combined with both the tracking and the tracing approach, provide a fine grained parallelism bound to a causal correct and optimized software development, focusing on efficiency and optimized performance for modern distributed systems.

The novel tracing approach helps users to reveal errors, problems and conflicts, improve system's performance, utilize limited resources more efficiently and facilitate development processes in a wide field of software domains. Compared to most commonly used trace formats, the described approach meets modern demands, constraints and requirements of distributed systems according to hard- and software issues such as memory accesses, cores, frequency or semaphores and timing metrics.

The comparison of various partitioning approaches reveals, that the proposed mechanism is capable of constraints and still preserves minimal runtime (number of SCSs) and minimal cross process communication in order to utilize parallel resources optimally. Various adaptions such as communication and computation cycle handling influences the mechanism and enables minimizing synchronization costs or waiting periods.

Applying the promising concepts to a program, the user benefits from an automatic partitioning and the assignment of execution units to any number of processes resulting in an optimal load balancing across processes and a trace, providing all necessary information for commonly used analysis or evaluation tools.

ACKNOWLEDGEMENT

The authors would like to thank Robert Preis for fruitful discussions according to the partitioning mechanism as well as the Amalthea consortium for a great exchange of different experience, knowledge and expertise focus.

6. REFERENCES

[1] Autosar – automotive open system architecture, January 2013, http://www.autosar.org.

[2] R. Preis, *Analysis and Design of Efficient Graph Partitioning Methods*, Ph.D. dissertation, University Paderborn, 2000.

[3] R. Hoettger, B. Igel, and E. Kamsties, A novel partitioning and tracing approach for distributed systems based on vector clocks, *Proceedings of the IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems, IDAACS'2013*, Berlin, 12-14 September 2013, pp. 670–675.

[4] L. Lamport, Time, clocks and the ordering of events in a distributed system, *Communications of the ACM*, (21) 7 (1978), pp. 558–565.

[5] F. Mattern, *Virtual time and global states of distributed systems, Parallel and Distributed Algorithms*, M. Cosnard et al. (editors), North-Holland, 1989, pp. 215–226.

[6] C. J. Fidge, Timestamps in message-passing systems that preserve the partial ordering, *Proceedings of the 11th Australian Computer Science Conference*, 1988, Vol. 10, pp. 56–66.

[7] A. Benveniste and G. Berry, The synchronous approach to reactive and real-time systems, *Proceedings of the IEEE*, (79) 9 (1991), pp. 1270-1282.

[8] Paulo Sérgio Almeida, Carlos Baquero, Victor Fonte, Interval tree clocks: A logical clock for

dynamic systems, *Proceedings of the 12th International Conference on Principles of Distributed Systems OPODIS'08*, Luxor, Egypt, *Lecture Notes in Computer Science*, Vol. 5401, 2008, pp. 259-274.

[9] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison Wesley, 1995.

[10] A. Nadgir and H. Haridas, Data flow partitioning schemes, 2003.

[11] Dataflow-oriented modeling with damos, March 2013, http://www.eclipse.org/proposals/tools.damos/.

[12] S. Peluso, P. Romano, F. Quagila, and L. Rodrigues, When scalability meets consistency: Genuine update-serializable partial replication, *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 2012, pp. 455–464.

[13] G. DeCandia, D. Hastorun, M. Jampani, et al., Dynamo: Amazon's highly available key-value store, *Proceedings of twenty-first ACM SIGOPS symposium on Operating Systems Principles*, 2007, pp. 205–220.

[14] L. Krawczyk and E. Kamsties, Hardware models for automated partitioning and mapping in multi-core systems, *Proceedings of the IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems, IDAACS'2013*, Berlin, 12-14 September 2013, Vol. 2, pp. 721–726.

[15] Y. Fong Lee, B. G. Ryder, and M. E. Fiuczynski, Region analysis: A parallel elimination method for data flow analysis, *IEEE Transactions on Software Engineering*, (21) 11 (1995), pp. 913–926.

[16] V. Elling and K. Schwan, Min-cut methods for mapping dataflow graphs, *Proceedings of the 5th International Euro-Par Conference on Parallel Processing, ser. Euro-Par'99*. Springer-Verlag, London, UK, 1999, pp. 203–212, http://dl.acm.org/citation.cfm?id=646664.701045.

[17] K. E. Schauser, D. E. Culler, and T. V. Eicken, Compiler controlled multithreading for lenient parallel languages, in *Proceedings of the 5th ACM Conference on Functional Programming Languages and Computer Architecture, Cambridge*, MA, USA, August 26–30, 1991, *Lecture Notes in Computer Science*, Vol. 523, 1991, pp. 50–72.

Robert Hoettger *received his B. Eng degree in 2011 in information technology and electrical engineering from University of Applied Sciences and Arts, Dortmund in Germany. He is a Master student and is going to start his Ph D studies in 2014. His research area covers distributed and parallel computing and model based software engineering with regard to automotive applications.*

Burkhard Igel *after study in electrical engineering and computer science received his doctoral degree (Ph.D.) in computer science from University of Dortmund. After more than 15 years working for Siemens Corporation now he is Professor at University of Applied Science and Arts in Dortmund and chairman of the supervisory board of item is AG. His research area covers distributed and parallel computing as well as requirements engineering and model based design.*

Erik Kamsties *is a professor for software engineering and embedded systems at the Dortmund University of Applied Science and Arts. He received a Diploma (M.S.) from the Technical University of Berlin and a doctoral degree (Ph.D.) in computer science from the University of Kaiserslautern (Germany). His current research focuses on requirements engineering, model-driven development, and embedded multi-core systems.*

computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

INTEGRATION OF CLOUD COMPUTING PLATFORM TO GRID INFRASTRUCTURE

Vladislav Falfushinsky, Olena Skarlat, Vadim Tulchinsky

V.M. Glushkov Institute of Cybernetics of NAS of Ukraine,
40 Acad. Glushkov Ave, Kyiv, 03179, Ukraine,
dep145@gmail.com, http://icybcluster.org.ua/eng/

Abstract: Both grid and cloud are used to organize large scale calculations and data processing on remote computers. Grid which became a basic computing infrastructure for the Large Hadron Collider experiments provides unified technical solutions for sharing and merging distributed heterogeneous computing resources within big collaboration groups. Cloud became popular among data centers and computing service providers because of flexibility, manageability and efficient hardware utilization. Both share common ideology "computing as a service", so one can expect additional benefits from their integration. The paper describes our approach to the integration. We propose to use cloud within grid sites for acceleration of application deployment and easy support of multiple virtual organizations by grid sites. The cloud in grid approach has been implemented and tested in Ukrainian National Grid, a part of European Grid Infrastructure.

Keywords: Grid computing; cloud computing; HTC; distributed computing; virtualization.

1. INTRODUCTION

Grid concept, architecture and middleware were developed by Ian Foster, Carl Kesselman and Steven Tuecke [1] since mid 1990th. But the wide application and popularity of grid in scientific calculations have been initiated by grid implementation of the Large Hadron Collider (LHC) data processing [2]. Instead of building a huge computer center to store and process 15 petabytes LHC data yearly [3], it was decided to organize their world-wide distributed storing and processing by the international collaborations of high energy physicists. The developed grid infrastructure and middleware appeared so powerful and flexible that many other researchers started to use it for their scientific calculations. European grid was organized in European Grid Infrastructure (EGI) and integrated with many national grids both within and beyond Europe [4].

Grid implements a paradigm of High-Throughput Computing (HTC) different from the High-Performance Computing (HPC) associated with parallel calculations [5]. When HPC targets minimal execution time for a job, the HTC aim is maximal utilization of all available computer performance to run multiple jobs or to solve a very big problem by parts. When a grid task is subdivided by subtasks the last ones are executed by different grid-sites according to their internal rules and queues without any guaranties of simultaneous run. It is the essence of distributed computing. Certificate based authorization, LDAP based resource catalogues and run environment specification languages create technical basis of grid.

Cloud proposes on-demand hardware usually with the pre-installed software. By sharing hardware of big datacenters cloud reduces the operational costs per task and increases efficiency of the hardware usage for the task flow [6]. It increases elasticity in environment selection and system mobility [7]. Typical requirements to cloud have been formalized by Peter Mell and Timothy Grance in [8]. Cloud providers offer virtualized computational resources with service-oriented provisioning and support "pay as you go" usage-based pricing model. The last feature is important for commercial clouds such as Amazon EC2, GoGrid, FlexiScale, and others.

In theory cloud gives the illusion that unlimited computing resources are available. Users can request, and are likely to obtain, sufficient resources at any time. In practice the illusion may be broken for large workloads or if the task is resource-dependent.

The users are allowed within both grid and cloud to acquire and release resources on-demand. Therefore, as the needs of the workflow change over

time, the releasing of the resources enables the workflow systems easily to grow and shrink their available resource pool. But specification of the requested resources in grid is implemented via its parameters. As the resource description is incomplete, the allocated nodes can be different, and nobody can guarantee their compatibility with the program. Cloud exposes resources with identical pre-installed system environments.

Cloud excels grid in benefits for workflow applications. Cloud applications are distributed in a common unified system and so can efficiently share data and exchange messages. Grid application sub-tasks can be separated both geographically and in time, so communication between them is tricky. Furthermore, cloud enables remote access to the allocated nodes, and thus supports on-line user-directed operations such as dynamic visualization.

Cloud can simultaneously run different operational systems on the same physical servers. So it provides more opportunities to combine independent software in solution of complex problems. E.g., data computed by a 32-bit Linux program can be than visualized in 64-bit Windows application within a common workflow.

Both grid and cloud propose ready to use computing environments on demand [9]. But cloud is less scalable and grid is less flexible and requires more administrators' efforts to fit contradictive system requirements of multiple virtual organizations. This is why integration of grid and cloud is in focus of modern researches.

The simplest kind of the integration is merging multiple geographically distributed computers in a single cloud based grid-site. Such way is proposed by the project "Experimental Deployment of an Integrated Grid and Cloud Enabled Environment in BSEC Countries on the Base of g-Eclipse" started in 2013 [10]. The project approach, however, is poor suitable for HPC tasks which need high performance parallel computations and use grid primary as a common access point to multiple clusters. However such approach can give more utilization power for big number of old hardware that can be used to virtualize nodes for some sorts of power HPC tasks.

More general approach is developed within the EGI-InSPIRE project established a Federated Cloud Task Force [11]. Some EGI grid-sites are already offering private cloud services for local research organizations. The project helps to unify their cloud architecture. Every site of the EGI Federated Cloud exposes the same programming interfaces for virtual machine setup and data manipulation operations, therefore applications that are built for one site of EGI Federated Cloud can run at any of the EGI cloud sites. In other words the EGI Federated Cloud provides an extensible set of reusable virtual machine (VM) images from the EGI VM marketplace. The images contain installed and set up scientific programs. The approach supposes complete virtualization of the EGI Federated Cloud grid-sites, which can decrease their HPC performance and prevent use of non-virtualized hardware (e.g. non-virtualized GPU accelerators). Besides, the whole grid resource pool is subdivided by two parts of cloud and ordinary grid-sites (Fig. 1).

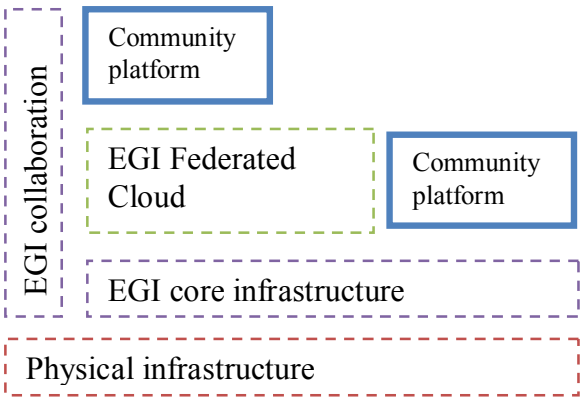


Fig. 1 – The EGI Platform Architecture.

Our idea is slightly different. We propose to run virtual machines as ordinary grid tasks and manage them similarly by grid tasks [12]. As result grid tasks can be run either in virtual or in physical environment. Thus routing problem solution can be accelerated by special efforts of either grid site or virtual organization administrators who install the necessary application software on physical servers. At the same time, a user can immediately run a special program version or configuration without its preliminary installation on multiple target grid-sites. He can easily add the preinstalled or own software to virtual machine, too. If the task is still running he can add some configuration issues to the virtual machine OS. Each user has its own set of virtual machines. In more restricted environment the administrator can control the list of available virtual machines for security reasons. If there is no appropriate template of virtual machine in repository, user can contact administrator and provide a preconfigured template.

In such architecture the cloud platform can be used also for secure testing the new software. Besides, it creates ability to allocate some resources for their online use (without queue) during the virtual machine grid task lifetime. Moreover there is an ability to make a shared access to some devices, that are installed on hypervisor i.e. accelerators, additional storage devices, cache etc. Such capability is absent in classical grid.

2. INTEGRATED PLATFORM ARCHITECTURE

The proposed integrated platform consists of 3 levels (Fig. 2):

1) the grid middleware responsible for the job delivery to appropriate clusters, for the data transfer, and for the user authentication;

2) the cluster scheduler responsible for the job queuing and distribution its subtasks between the cluster nodes;

3) the cloud platform software responsible for managing the private cloud as the cluster virtual part.

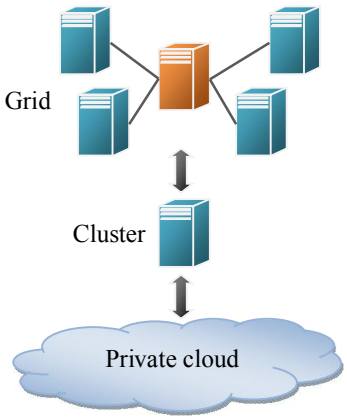


Fig. 2 – The cloud platform integrated in the grid infrastructure.

The grid user generates a proxy-certificate for an appropriate lifetime period, creates the job specification as a file (e.g. in the xRSL format) and launches the job using the grid submit command. The grid scheduler directs the job on a proper grid-site according to the job resource specifications. If the job specification contains links to files, grid automatically downloads them from user's local file system, FTP, HTTP, the grid storage or other place and puts them to a folder or a block device which is shared to the virtual machine file system. The cluster scheduler accepts the job, inserts its command into the queue and runs according to the queue rules. The integrated cloud platform does not affect the grid workflow.

Since the virtual systems need unified path to the input files, the calculation results, and the intermediate files which are download/uploaded by the grid middleware during the job submitting, the integrated platform is based on the unified storage structure rules. Within the Ukrainian National Grid (UNG) [13] all grid application files are managed according to the next rules:

- each grid user accesses UNG resources only under a certificate associated with a Virtual Organization (VO);
- each VO has its own directory in the grid storage;
- the filenames of system files and folders in the VO directory are started from dot (to be invisible for users);
- each VO directory contains system folders ".apps" and ".cloud" for user applications and backups of the system images;
- other subdirectories of the VO directory are shared to all members of the VO.

The cloud platform main unit called *instance* is a virtual node of the cluster. Each instance is created from one of pre-installed VM images. The instances are managed by users and VO administrators via a set of cloud management commands. The commands can be submitted as ordinary grid jobs, according to usual grid and cluster security rules and restrictions, which can deny users direct access to the virtualization system. The cloud management command set is supported by management components, the core of the integrated cloud platform.

Among them there is a routing service which opens a certain port for remote access to the instance through Remote Desktop Protocol (RDP). This is a key to extension of traditional grid functionality by dialog user interface capabilities. RDP remote desktops can be used from personal computers and mobile devices such as smartphones or tablets. Besides, RDP simplifies downloading/uploading files during the instance lifetime. (This is important for long time executed dialog programs, as the traditional grid tools wait for a job finis to download its results.)

Speaking more, the management components create the most important part of the intermediate layer between the private cloud and the grid. Other necessary parts include a virtualization server and a database of VM images & instances, constraints, the virtual network settings, the software licensing information and so on (Fig. 3).

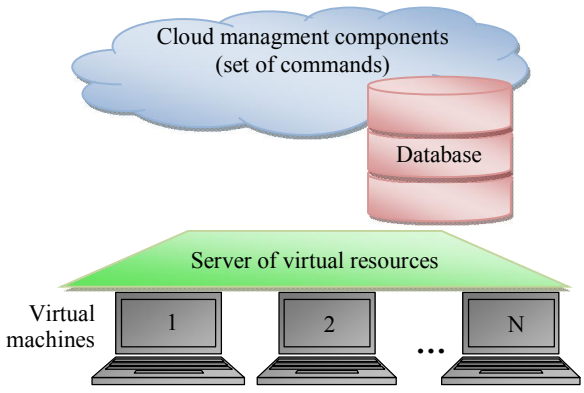


Fig. 3 – Intermediate layer between the cloud platform and the grid.

The list of cloud management commands for VM control in the integrated grid infrastructure includes:

- `VMmanager` to list the images, to list, create, start and shutdown the virtual nodes;
- `VMRunApp` to execute a user command within an instance (on the corresponded virtual machine);
- `VMRegisterApp` to install an application in an instance;
- `VMDBCreate` to test the system, create and restore the cloud platform DB. (The command permissions may be restricted to administrators only);
- `VMAddImgTpl` to manage the instance templates which relate the instance parameters to the physical node parameters.

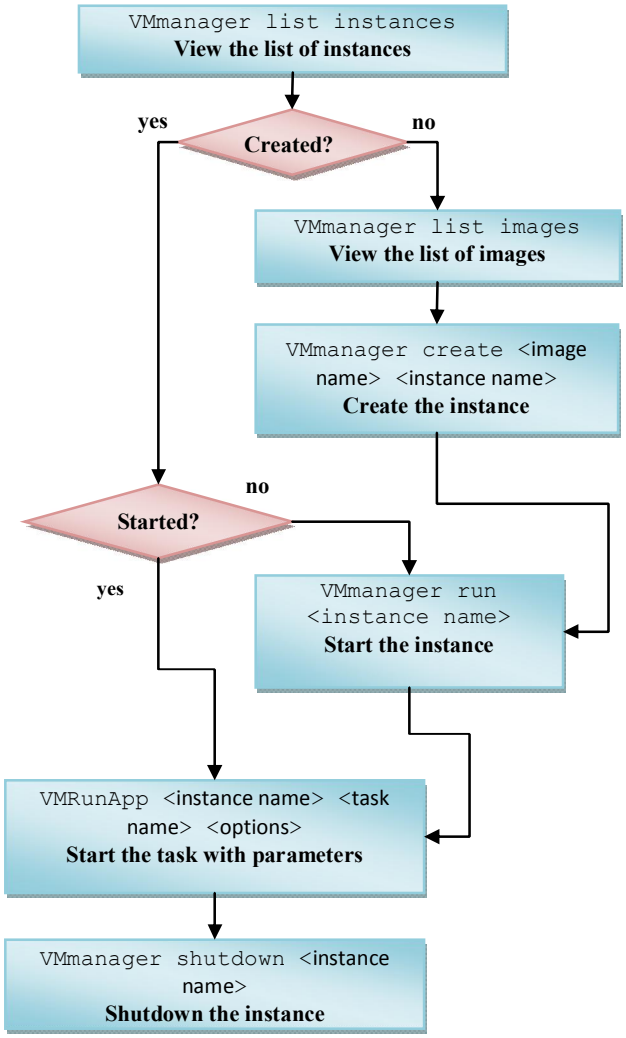


Fig. 4 – Cloud management commands in use.

The cloud management commands application can be illustrated by the next scenario.

At first, the user inspects the list of the available instances by VMmanager. If the requested instance is not running, it can be started by

```
VMmanager run <instance name>
```

After successful launching the instance the command

```
VMRunApp <instance name>
  <task name> <parameters>
```

is called to run the task with parameters on the specified instance. In order to free allocated resources, the instance can be stopped by

```
VMmanager shutdown <instance name>
```

If the requested instance is absent it can be created from an image. The list of images can be retrieved by VMmanager list images. The new instance is created by

```
VMmanager create <image name>
        <instance name>
```

The cluster administrator can call management commands for managing templates and cloning images. The whole scenario can be described as one grid script for the integrated cloud platform (Fig. 4). After the script is started the user is allowed to acquire additional connection options (like RDP address, port, user and password) calling the status command from grid utilities.

3. INTEGRATED PLATFORM IMPLEMENTATION

Core components of clouds are VM hypervisors which host all the VM instances. The integrated platform accesses hypervisor via the opensource toolkit libvirt [14]. This library acts as a middleware between hypervisor and cloud management components. It is compatible with most popular hypervisors, supports recent Linux versions and provides modern set of virtualization capabilities:

- management of virtual machines, virtual networks and storage;
- portable client API for Linux, Solaris and Windows;
- remote control with authentification and encryption;
- unified management of multiple hypervisors from one access point;
- a driver-based architecture and common hypervisor-independent API;
- integrated services: HTTP, DHCP server, VPN, SSH, built-in shell.

The recommended VM hypervisor tested in our platform is Oracle VirtualBox [15]. The choice was motivated by its low resource consumption, fast configurability, multi-system VM (Linux, Windows and MacOS), and parallel run of multiple VM containers. The licensing policy favorable for Academia also affected the decision.

Special efforts targeted user authorization and authentication in the integrated platform. It is important to keep the extended access capabilities within the standard grid rules and use nothing but

the time-limited proxy-certificates of grid users. The proposed solution is based on the proxy sharing via Myproxy Credential Management Service [16]. It combines the online credential repository with the online certificate authority, and allows users to obtain securely credentials when and where needed. The system is configured to encrypt all private keys in the repository with user-chosen passphrases and server-enforced policies for passphrase quality. The technique also provides delegation of the credentials from one user to another without using certificate files and its passphrases.

To put the integrated platform in operation the web user's interface (Fig. 5-7) has been developed.

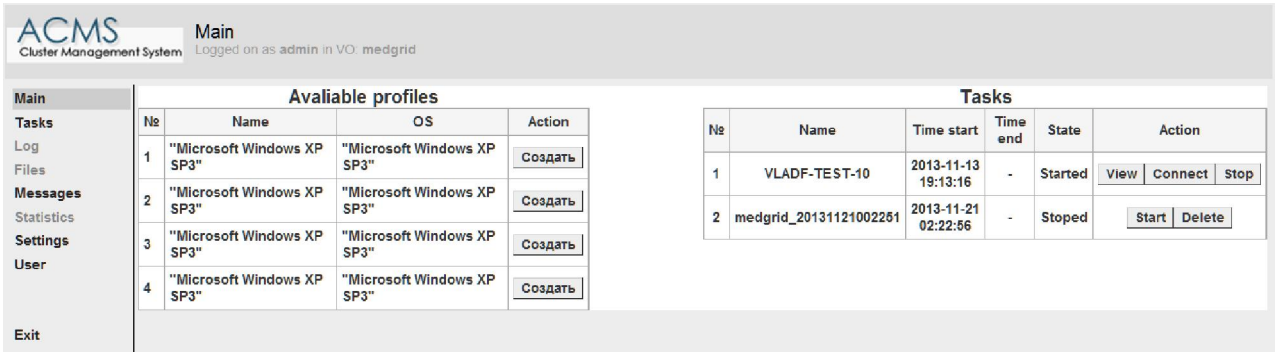


Fig. 5 – Web interface of the integrated platform.

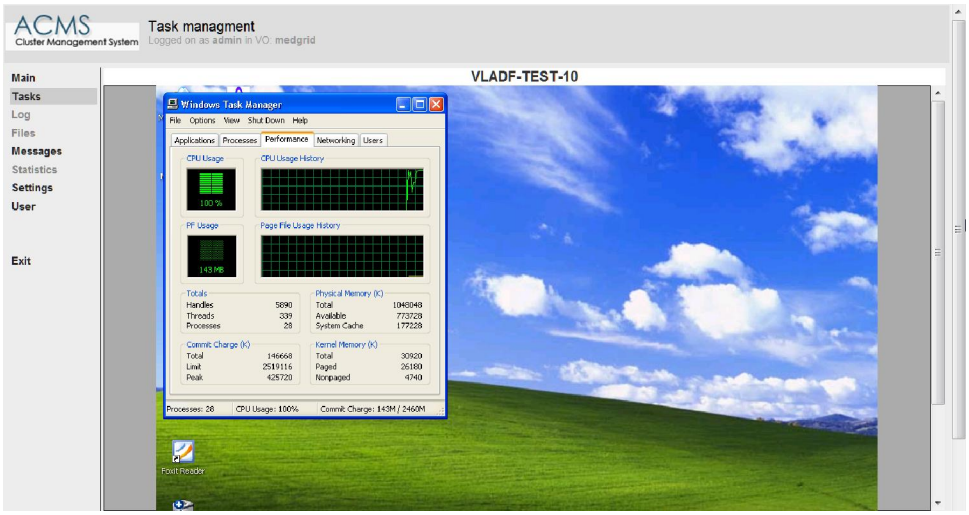


Fig. 6 – Windows XP run as a VM task on Linux cluster via ARC grid middleware.

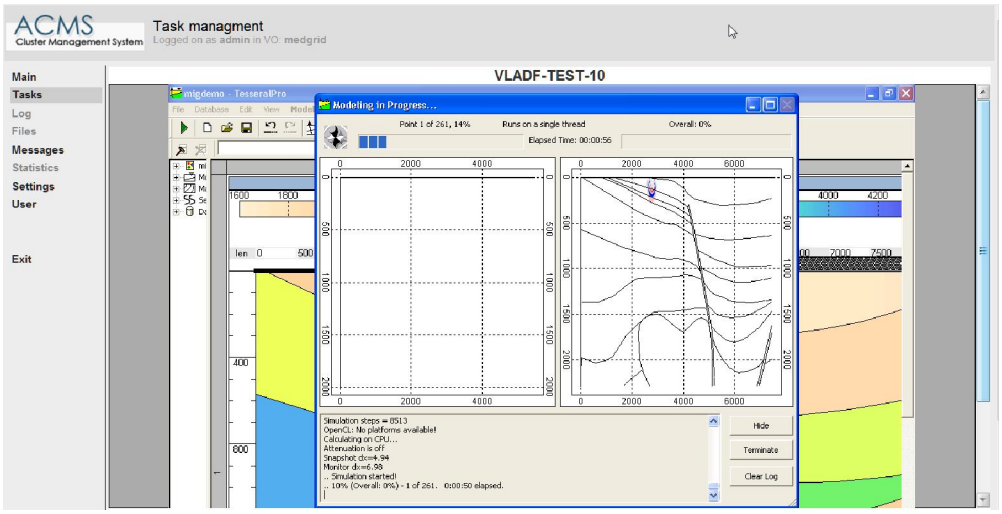


Fig. 7 – Seismic modeling program Tesseral Pro for Windows executed in grid in the integrated platform and available from the web interface.

First end-user experience of application the integrated platform in VO "Medgrid" [17] and "Geopard" [18] resulted in improvements and development of additional functions:

1. Delayed start of grid tasks. The grid tasks related to cloud management are submitted in background to unfreeze the task manager during the long enough procedure.

2. Accelerated work with remote grid storage. Virtual copy of the VO folder directory is stored in database. In addition to acceleration of usual operations it helps to synchronize or re-install local storage. Current recovery speed for all directory structure is about 1000 files per minute. The synchronization tool is run periodically in background.

3. Cache of VO user certificates was added to web interface. This function allows to read statistics and to communicate with other VO members without generating proxy certificates. (Access to grid is not possible.)

4. PHP API for ARC implementation of the integrated platform with basic functions of monitoring, submitting, interruption and deleting tasks, file operations in grid storage.

5. Web interface API for remote administration of the web interface allows easy integrate the solution in another management system.

4. CONCLUSION

The offered integration approach unites the principles of both grid and cloud computing. The described implementation of the cloud platform integration in grid-infrastructure has been tested and applied in operation within Ukrainian National Grid which is an integrated part of EGI. UNG is partially based on ARC middleware. But the approach is suitable for gLite and EMI grids as well. The developed set of commands is sufficient for flexible command line interaction with the integrated in grid cloud platform. Main advantages of the proposed solution are:

1) quick deployment of new or alternative program versions within VO, less administrator's efforts,

2) arbitrary mix of grid and cloud/grid tasks on the same clusters,

3) dialog and on-line environments run in grid for immediate user's operations (to exclude delays for submitting and re-submitting grid tasks),

4) automated data flow and distributed storing.

5) Linux/Windows portability,

6) tolerance to differences in operational environments and VM hypervisors of different grid sites.

5. ACKNOWLEDGMENT

Authors express gratitude to Andrii Salnikov, Ievgen Sliusar and Oleksandr Boretskyi from Taras Shevchenko National University of Kiev for development of VM task management from ARC and for configuration of the network infrastructure. The development could not be finished without their help and efforts.

We also thank State Technical Research Program of Development and Application of Grid Technologies for support, Inparcom [19] for the hardware platform of the development, and "Uber-Cloud Experiment" [20] for the great experience.

6. REFERENCES

[1] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid enabling scalable virtual organizations, *International Journal of High Performance Computing Applications*, (15) 3 (2001), pp. 200–222.

[2] M. W. Browne, Physicists discover another unifying force: Doo-Wop, The New York Times, April 2008, pp. 4–6.

[3] R. Lucio, Superconductivity: its role, its success and its setbacks in the Large Hadron Collider of CERN, *Superconductor Science and Technology*, (23) 3 (2010), pp. 17-21.

[4] European Grid Infrastructure, http://www.egi.eu

[5] J. Schiff, Grid computing and the future of cloud computing, *Proceedings of the Enterprise Storage Forum*, January 21, 2010.

[6] R. Bias, Grid, Cloud, HPC … What's the Diff?, http://www.cloudscaling.com/blog/cloud-computing/grid-cloud-hpc-whats-the-diff/, November 18, 2010.

[7] J. Myerson, Cloud computing versus grid computing, http://www.ibm.com/developerworks/library/wa-cloudgrid/, March 03, 2009.

[8] P. Mell, T. Grance, Special Publication 800-145 The NIST Definition of Cloud Computing, *National Institute of Standards and Technology*, September, 2011.

[9] I. Foster, Yong Zhao, I. Raicu, Lu Shiyong, Cloud Computing and Grid Computing 360-Degree Compared, *Proceedings of the Grid Computing Environments Workshop*, 2008, pp. 1-10.

[10] Experimental deployment of an integrated grid and cloud enabled environment in BSEC countries on the base of g-Eclipse, http://blacksea-cloud.net/

[11] Federated Cloud, http://www.egi.eu/infrastructure/cloud/

[12] Vladislav Falfushinsky, Olena Skarlat, Vadim Tulchinsky, Cloud computing platform within grid infrastructure, *Proceedings of the 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2013)*, Berlin, Germany, 12-14 September 2013, pp. 717-720.
[13] Ukrainian National Grid, http://ung.in.ua/
[14] Libvirt, http://libvirt.org/
[15] Oracle VirtualBox VM, https://www.virtualbox.org/
[16] MyProxy Credential Management Service, http://grid.ncsa.illinois.edu/myproxy/
[17] Medical Grid-system for population research http://medgrid.immsp.kiev.ua/secondary_pages/home_en.html
[18] UNG Virtual Organization for grid applications in Earth sciences, http://vo.igmof.org.ua/ (in Ukrainian)
[19] Intelligent computers «Inparcom», http://www.inparcom.com
[20] Uber-Cloud Experiment, http://hpcexperiment.com

Vladislav Falfushinsky, *Junior researcher and PhD student at V.M.Glushkov Institute of Cybernetics of NAS of Ukraine. Master of Computer Science: Intelligent systems for decision making, National University of Kyiv-Mohyla Academy, 2010. Interests: grid, cloud, large scale and distributed software systems, parallel application software (primary for geophysics), databases and big data processing.*

Olena Skarlat, *Junior researcher at V.M.Glushkov Institute of Cybernetics of NAS of Ukraine. Master of Computer Science: Information Control Systems and Technologies, National University of Kyiv-Mohyla Academy, 2010. Interests: Analysis of the Ukrainian regulatory framework and modelling business-processes for e-Services – Software assessment – ISO and ETSI standards harmonization– Long-term preservation strategies– Adopting grid-technologies for investigations.*

Dr. Vadim Tulchinsky, *Head of High-Performance and Distributed Computing Lab at V.M.Glushkov Institute of Cybernetics of NAS of Ukraine. Interests: parallel and distributed computing; big data, seismic data processing, databases; simulation, computational methods in geophysics.*

computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

HARDWARE MODELS FOR AUTOMATED PARTITIONING AND MAPPING IN MULTI-CORE SYSTEMS USING MATHEMATICAL ALGORITHMS

Lukas Krawczyk, Erik Kamsties

Dortmund University of Applied Sciences and Arts
Emil-Figge-Str. 42, 44227 Dortmund, Germany
E-mail: lukas.krawczyk@fh-dortmund.de; erik.kamsties@fh-dortmund.de

Abstract: Multi-core CPUs offer several major benefits in embedded systems. For instance, they usually provide higher energy efficiency and more computing power compared to single-core CPUs. However, these benefits do not come for free: A program has to be divided into tasks, which can be executed in parallel on different cores. Partitioning of software and mapping on cores are nontrivial activities that require detailed knowledge about the underlying hardware platform, e.g., the number of cores, their speed, available memories, etc. Such information is typically stored in handbooks. If this information would be available in a machine readable model, we call it hardware model, the partitioning and mapping activities can be automated. In this paper, we propose a hardware model and illustrate it using an example of a Freescale multi-core CPU. We then discuss a small case study situated in the automotive domain, which illustrates the use of the hardware model in partitioning, mapping, and code generation.

Keywords: Multi-core; hardware model; embedded systems software development; target mapping; model-driven development; Autosar.

1. INTRODUCTION[1]

The demands on mobile and embedded systems are ever increasing. Mobile phones offer strong multi-media capabilities and, for instance, embedded systems in cars implement image recognition to analyze radar images of traffic in an adaptive cruise control. Mobile and embedded systems benefit in several ways from multi-core CPUs. These CPUs provide more computing power at the same clock speed resulting from several cores working in parallel. They provide better energy efficiency because they run on a lower clock speed compared to a single-core with the same computing power and cores may be switched off if their power is not needed. Furthermore, multi-core CPUs allow for high-assurance systems by running two cores redundantly in a so called lockstep mode.

A program which utilizes the benefits of a multi-core CPU has to be divided into a set of communicating tasks, which can be executed in parallel without blocking each other because of synchronization on shared resources. In order to find an optimal partitioning and mapping, hardware-related information must be taken into account. A trivial example is the number of cores, more advanced information includes the type and speed of shared memories.

Such information about a CPU is typically stored in large processor handbooks. If hardware information would be available in a *machine readable* model, we call it hardware model, the partitioning and mapping activities can be further automated.

In this paper, we propose a hardware model which is rich enough to describe systems of heterogeneous multi-core CPUs as well as peripheral hardware. The use of the hardware model in partitioning and mapping is shown in a case study. Additional application for code generation is outlined. Furthermore, we provide an example of a hardware model for a Freescale MPC5668G multi-core SoC popular in the automotive domain.

This paper is organized as follows: Section 2 discusses the related work. Section 3 outlines the hardware model and the example model is shown in section 4. The main part of this paper is a case study on how hardware models support partitioning, mapping, and code generation, followed by a simple

[1] The research leading to these results has received funding from the Federal Ministry of Education and Research (BMBF) as part of the AMALTHEA project.

Yakindu DAMOS[2] based example illustrating the respective steps of the case study for an automated partitioning and mapping of automotive software in section 6. A conclusion and directions for future work close this paper.

2. RELATED WORK

The idea of utilizing models to support particular steps of development has been pursued for years. For instance, the probably best known application of hardware models is hardware synthesis, which allows transforming a given formal description of hardware into an implementation. This topic is being performed and researched since decades and lead to the introduction of several hardware modeling languages, such as SystemC, VHDL, SystemVerilog etc. [1] and tooling which utilized these languages. Hardware models usually describe the structure and behavior of hardware at a high abstraction level e.g. in terms of register-transfers (VHDL). The true hardware models used by the chip manufacturer for hardware synthesis are considered as intellectual property (IP) and thus usually not publicly available.

Hardware models with an even higher abstraction level have been described in EAST-ADL [2] and AUTOSAR [3]. They are used within the development of automotive embedded systems and support preliminary allocation decisions and the configuration of micro controllers. Compared to these types, we need a hardware model which is located in between: Common hardware description languages like SystemC are still far too detailed to support our cases, and important implementation details, like the cycles per second, are yet unknown. On the other hand, AUTOSAR and EAST-ADL are not specialized enough to provide the required amount of information.

An algorithm for the automatic partitioning and mapping of embedded software for a *homogenous* multiprocessor system has been described, among others, by Cordes et al [4]. It is based on integer linear programming and utilizes a model of the hardware platform with information of its communication and task-creation overheads. Our hardware model targets to support algorithms for *heterogeneous* multi-core system partitioning and mapping, hence much more hardware related information is required. This includes, but is not limited to, the specification of unique characteristics of the cores, e.g. a FPU, as well as additional constraints which will restrict mapping decisions.

Our previous publication [5] is dealing with the partitioning and mapping for heterogeneous multi-core systems using a hardware model. It outlines a pragmatic approach for partitioning and mapping of data flow graphs with a simple algorithm. In this paper, we seek to determine the *optimal* allocation of tasks to cores in due consideration of allocation constraints. As such, we need a more versatile approach, e.g. integer linear programming, which has to be supported by the hardware model.

3. HARDWARE MODEL

The purpose of our hardware model is the support of embedded systems development in general, i.e. regardless of the embedded systems field of application (automotive, mobile, …). Common steps within this procedure are usually partitioning and mapping of embedded software as well as code generation. One of our goals is to provide compatibility with AUTOSAR, which is the current standard in the automotive domain.

The meta-model of our hardware model is shown in Fig. 1. Classes shown in the upper left corner represent the three main hierarchies of elements that may be modeled: *Components*, *Ports* and *Pins*. This structure is oriented at AUTOSARs ECU Resource Description and allows a direct mapping between both models elements. Classes in the upper right corner represent the data-types additional attributes of the elements may take (e.g. *Boolean*, *Integer*, *Long*, ...). Classes in the lower left corner represent the extensions that have been made to enhance the amount of information compared to AUTOSARs ECU Resource Description and as such allowing us to utilize the model beyond the automotive scope.

We introduced a hierarchy for descriptions up to system level, allowing describing a system of ECUs, each consisting of an unbound number of System-on-Chips (SoCs) which may contain multiple cores. Each element may operate on a different frequency which is described by a *Prescaler* and its referenced *Quartz*. The Memory class is used to describe any type of memories on different hierarchies, e.g. a cache as well as RAMs or ROMs. The *Network* class is used to span a network which is accessed through *ComplexPorts*. This allows deriving a memory map out of the hardware model which supports the detection of concurrent access to any type of component.

4. HARDWARE MODEL EXAMPLE

One of the domains which favors the usage of heterogeneous multi-core CPUs is the automotive domain, which is the reason why our example focuses on this branch. Yet the hardware model itself is not limited to this domain.

[2] http://blog.yakindu.org/category/damos-2/

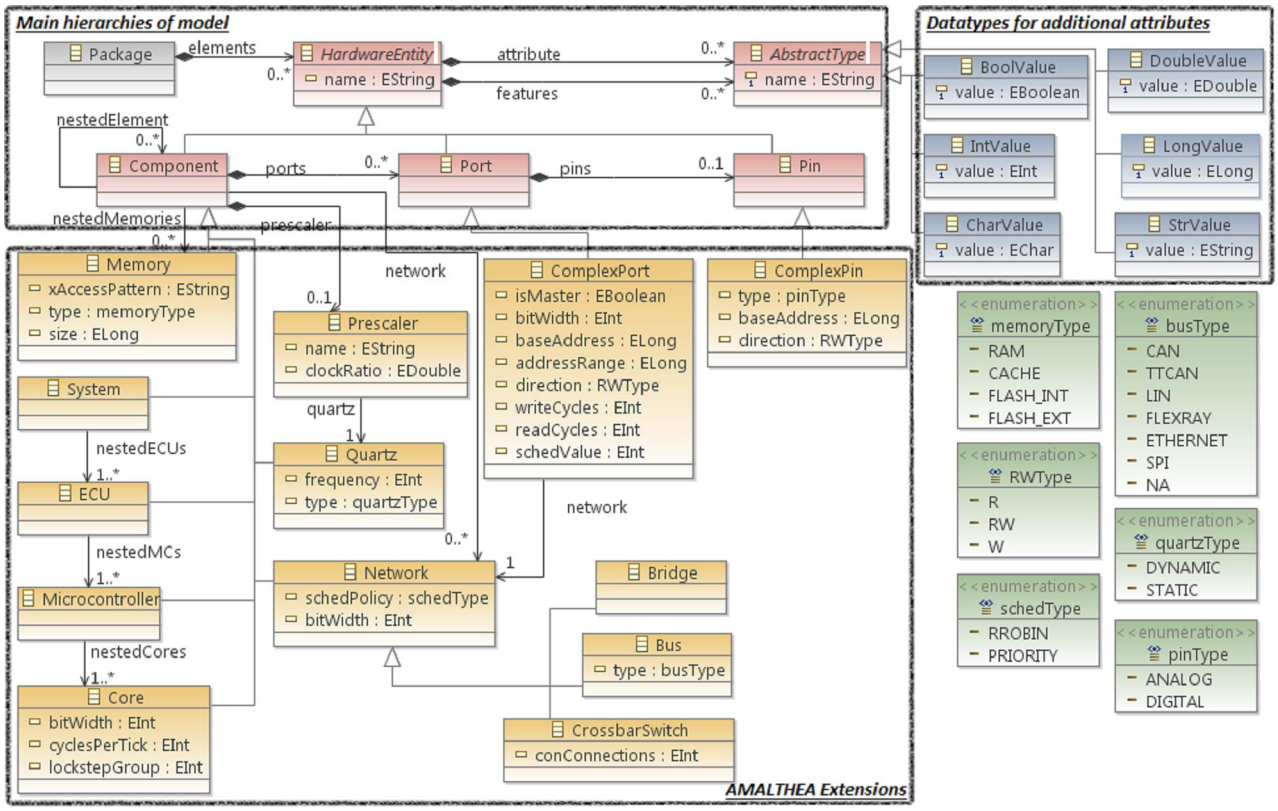


Fig. 1 – EMF based meta-model describing the hardware model.

The simplified hardware model based on the heterogeneous Freescale MPC5668G multi-core SoC is illustrated in Fig. 2. In this figure, blocks in the upper row represent hardware components with master access on the network while the blocks in the lower row indicate slaves. Networks on the SoC which are referenced by Ports (white rectangles) and peripheral elements are represented by blocks in the mid row.

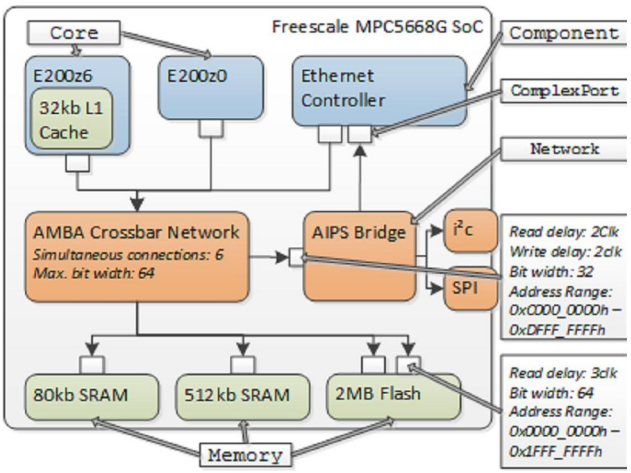


Fig. 2 – Simplified illustration of the MPC5668G SoC hardware model.

The Freescale MPC5668G SoC contains two heterogeneous cores. The main core is the e200z6, which operates at 116 MHz, has 32KB L1 cache and supports floating-point computations. An additional e200z0 core, operating on half the e200z6 frequency, is available as I/O processor. The memory consists of 2MB flash and 592kb RAM which is split up into one 512kb and one 80kb module to allow concurrent access by the masters. The network, which is provided by the AMBA Crossbar Switch, allows up to 6 concurrent connections from masters to slaves with up to 64 bit width. Further interfaces, like I²C and SPI, can be accessed through an AIPS Bridge.

5. USAGE OF A HARDWARE MODEL IN PARTITIONING AND MAPPING

The case study within this paper targets at the hardware model support of the steps which are required to partition and map software to multiple cores and generate target specific ready to compile code. To achieve this, we follow Fosters PCAM methodology for designing parallel algorithms [6]. His methodology specifies the steps partitioning, communication analysis, agglomeration, and mapping. Code generation follows Herrington [7].

5.1. PARTITIONING

The first step is the decomposition of software models, which involves to determine the task granularity and which computations should be part of a coherent set. According to Foster, this step is

intended to reveal parallel execution opportunities of a problem by partitioning it into fined-grained decompositions, providing the greatest flexibility for parallel algorithms. However, it should be avoided to replicate data or computations, e.g. both should form disjoint sets.

This step requires information about peripheral elements with their base addresses and address ranges (e.g. a memory map). This allows determining which addresses belong to one specific periphery and which tasks can be merged (as they address the same piece of hardware) or split (as they access different/independent entities of hardware). Based on this information, a partitioning and mapping algorithm is able to analyze software models in consideration of a specific target platform and to decompose tasks in a target optimized manner.

5.2. COMMUNICATION ANALYSIS

The second step is the communication analysis. Once a model has been partitioned into tasks, data dependencies between the respective tasks will have evolved. For instance, a former coherent process might now be split up into multiple processes, with each of them depending on the results of the respective other process.

In communication analysis, such dependencies are identified and the inter-task communication as well as its cost is analyzed. This step has two phases: The first phase involves the definition of a channel structure. Each channel links two tasks and allows the communication between tasks that require data and the respective possessors of these data. In the second phase, the messages which are being communicated on these channels are derived and defined.

To support this step, we need information about data type implementations. As it is well known, the size of data types usually depends on the target operating system and compiler. For instance, data types like *long*, *double* and *int* may have several valid implementations which differ in their size and, as such, affect the communication overhead. Depending on the concrete implementation, the number of transferred bytes by an *int* type variable may vary between 2 and 8 bytes.

5.3. AGGLOMERATION

After an initial set of tasks has been specified and communication dependencies between these tasks identified, it is required to agglomerate the tasks into greater task sets. In the agglomeration stage previous decisions are revisited and the tasks further optimized towards a parallel platform. This may be achieved by simply merging decomposed tasks into one or more greater tasks, for instance, if the tasks have a too fine granularity for a specific underlying hardware platform with a high task creation overhead. Another aspect in the agglomeration step is the replication of computations and data.

To support this step, information about the number of cores, the communication channels and available memories as well as processor cache are required. An agglomeration algorithm will consider cache sizes that will have a significant impact on the decision if data replication should be applied or not. Furthermore, the available capacity of the communication channels of a specific target platform steers the granularity of the agglomerated tasks, e.g. slow channels favor fewer tasks while a high-speed on-chip network could handle even many tasks of fine granularity.

5.4. MAPPING

The mapping step consists of the allocation of software model parts to elements of a hardware platform. The purpose of this step is to specify which task should run on which processor of the target platform, providing the target platform is a multi-core system without automatic task scheduling. The goal of the mapping algorithm is to minimize the execution time by:

- Increasing concurrency, i.e. distributing tasks on different processors.
- Increasing locality, i.e. arrange tasks which communicate frequently on the same processor.

Our hardware model contributes towards this with specific information about cores and their parameters, such as frequency, their target application etc. In addition, a generic possibility to define constraints is provided, as these take a vast variety of options, such as the maximum number of processes, required instruction sets or safety-constraints.

The communication between execution units is the second aspect which has to be taken into account. Naturally coherent computations may only be distributed between interconnected executional units. Additionally, the communication paths between these units may contain several constraints, prohibiting several constellations. It is essential to know these as well as the capacity of the routes. To achieve this, our hardware model provides basic information about the complete network structure of the target system, regardless of the abstraction level its implemented (e.g. network of embedded systems, one specific embedded system or merely a SoC. / micro-controller). The information contains details about any type of map-able network on an abstract level (e.g. the networks speed, address space,

scheduling policy etc,) as well as what participants are connected to it. In addition, a generic structure for further constraints is allowing an optimized mapping, e.g. to ensure reliability by safety constraints.

5.5. CODE GENERATION

The final step is code generation for a specific target. Having the tasks and their mapping specified, all required information for a code generator is available and code generation may be performed. Our scope is to develop ready to compile code for a specific hardware, also known as platform dependent code generation.

Two approaches for code generation are available. The first approach is code generation for abstract interfaces. Usually an API for the access to the underlying hardware platform, e.g. a *Hardware Abstraction Layer (HAL)*, is introduced and code utilizing this API generated. However, this approach has several downsides. On the one hand, the complexity of the API to be implemented has to be estimated. An API with little functionality may be realized very fast but will lack in efficiency and/or flexibility. On the other hand, an implementation with a wide scope of functions will be time consuming and only be worth if the platform is used in multiple projects. A tradeoff between these granularities has to be predicted, which is not always possible.

Parameterization of the code generation is the second approach. Here, rules and/or templates are used to customize the code generator for a specific target platform. Templates may be further refined with macros which are replaced by additional hardware related information or code. As it may be considered that the code was transformed correctly, it is unlikely that further maintenance operations by users are necessary. This allows mixing application specific code and target platform specific code, permitting the compiler to perform common optimization techniques. Usually a typical compiler includes a mixture of both approaches, i.e. has several parameterized parts and accesses manually written platform dependent code through a specified interface.

Regardless of the actual approach, the amount of information to support this process is the same. While attributes of certain elements (e.g. ports and pins, registers and memories, data path addresses etc.) provide structural information which is replaced by the code generator, code templates and/or snippets allow to complete more challenging tasks, like the initialization of specific controller or even implement the hardware dependent layers of a messaging protocol with its according functions.

6. CASE STUDY

This section briefly describes the experimental environment for an integer linear programming (ILP) based approach for partitioning and mapping software for an embedded system utilizing the hardware model. A more detailed description of the resp. steps within the tool flow is given by the following subsections. The software is represented by the Yakindu Damos data flow model in Fig. 3 which describes a simple cruise control unit.

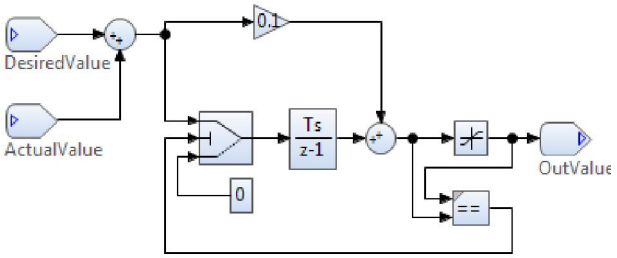


Fig. 3 – Example Data Flow Model

The structure of the steps which are performed with the tool chain in our experimental environment, which has been developed and implemented within the itea2 project AMALTHEA, is shown in Fig. 4.

Its first tool is the *hardware aware partitioning tool*, which will perform the steps partitioning (a) and communication analysis (b) with regard to a chosen hardware platform and pass the resulting model to a so called *graph partitioning tool*. This tool will divide the graph into smaller sub-graphs, which technically equals the agglomeration (c) of smaller executable units into tasks (i.e. each sub graph represents one task). The next step is the mapping (d) which is performed by an ILP based *mapping tool*. In the final step, two *code generators* produce the target platform code (e).

6.1. PARTITIONING

Yakindu DAMOS is capable of extracting a so called *Execution Graph* (i.e. a cycle free graph of the Data Flow Model) which serves as input for our approach. This model has already a very fine granularity, therefore we are able to skip any further decomposition of the model and focus on splitting and merging blocks which are using shared hardware components.

In this example, the blocks `DesiredValue` and `ActualValue` are reading data from the `I²C` and `SPI` interfaces. As shown in the Freescale MPC5668G SoCs hardware model (Fig. 2), these interfaces are located behind the `AIPS Bridge`, therefore it is wise to merge them in order to reduce overheads which might occur by task creations or

context switches and prevent mutual exclusion (Fig. 4(a)).

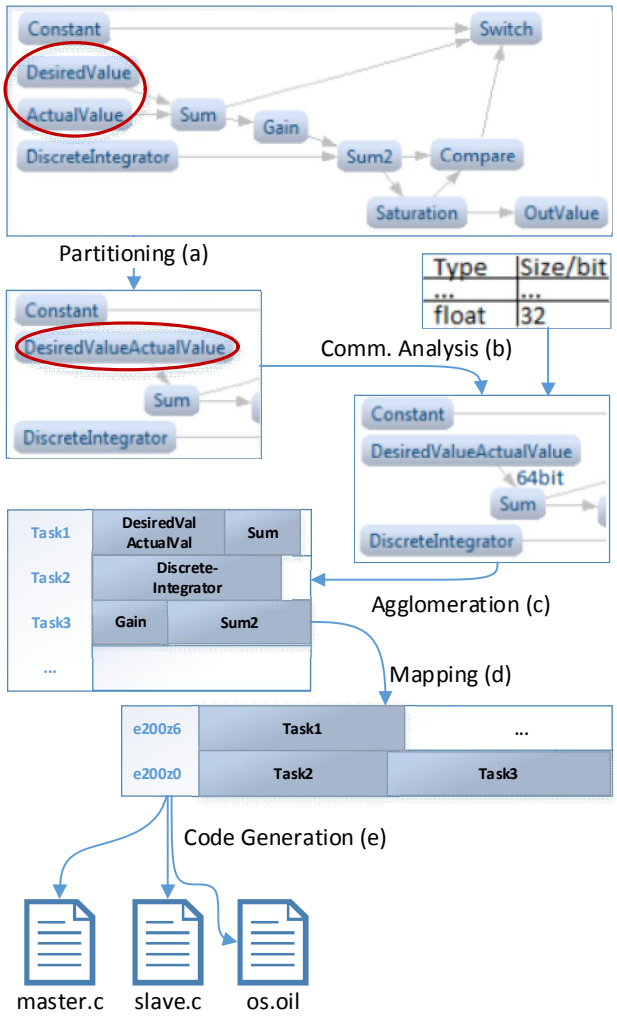


Fig. 4 – Partitioning and mapping approach.

6.2. COMMUNICATION ANALYSIS

The communication cost of the model can be determined by the number of data transfers between the blocks and their respective data type size. As usually several operating systems are available for a specific hardware, the information about the concrete implementation is stored in tables that are attached to the hardware model. This allows to calculate the communication cost and enhance the edges of the execution graph with these values (Fig. 4(b)), providing the required information for the following steps.

6.3. AGGLOMERATION

The agglomeration is performed by a graph partitioning algorithm based on vector clocks (see [8]). This algorithm is implemented in a *graph partitioning tool* which merges the Blocks from the DFG model into larger groups of tasks. The hardware model supplies information about (i) the maximum number of simultaneously executable tasks (i.e. cores, threads per core, …) as well as (ii) the task creation overhead. This allows steering the granularity of the resulting tasks resp. the maximum number of tasks to create. The graph from the previous step describes the relation between the blocks as well as their coherence, which has impact on the sorting order of the Blocks as well as decision which Blocks will be distributed into the which task (Fig. 4(c)).

6.4. MAPPING

The mapping (Fig. 4(d)) in this algorithm is performed by a pragmatic ILP (Integer Linear Programming) approach based on [9], which focusses on minimizing the maximum execution time of concurrently operating cores. The mapping tool utilizes the open source oj!Algorithms[3] project which we use to solve the ILP equations.

The first step in this mapping algorithm is to determine the required execution time $ET_{i,j}$ of each task i for the resp. core j. This is can easily be determined by (1) and (2).

$$ET_{i,j} = CT_i * CC_j, \tag{1}$$

$$CC_j = TC_j * PS_j * Q_j, \tag{2}$$

where CT_i is the number of cycles which are required to process the task and CC_j the number of cycles the core can process per second. The variable TC_j describes the number of ticks that are required to process one cycle, PS_j the prescaler for frequency scaling and Q_j the frequency of the quartz attached to the core.

The second step is the formulation of mapping constraints e.g. to limit the number of cores a task will be allocated to one core (3)

$$\sum_{j=1}^{n} A_{i,j} = 1 \quad \forall i \in [m], \tag{3}$$

with $A_{i,j}$ describing the allocation of task i to core j, m number of tasks and n number of cores.

This equation however is only valid, if all cores are suitable to process this task. As some of the tasks may contain special requirements on a core, e.g. the presence of a Floating Point Unit (FPU) or a specific instruction set, it would be also required to narrow

[3] See http://ojalgo.org/

down the solution space to valid cores. In our example for instance, only the main core *e200z6* contains a FPU, hence we would need to limit the scope of valid cores to this core only.

A general formulation to narrow down the solution space for multiple valid cores is given in (4)

$$A_{i,j} = 0 \quad \forall j \notin V \,, \tag{4}$$

with V being a group of valid cores.

A very simple approach to minimize the execution time can now be achieved solving the equations (5) and (6) as mentioned in [9]

$$z \rightarrow \min \,, \tag{5}$$

$$\sum_{i=1}^{m} A_{i,j} * ET_{i,j} \leq z \forall j \in [n] \,, \tag{6}$$

with z being the maximum execution time of all concurrently executed cores.

6.5. CODE GENERATION

The code generation for this algorithm is performed by two code generators (Fig. 4(e)).

The first code generator is provided by Yakindu DAMOS and innately capable of producing hardware independent code for the resp. blocks of the data flow graph. To support target ready code generation, hardware related information is provided by the hardware model. This consists of libraries for I²C and SPI access as well as the addresses of hardware components, i.e. the memories and peripherals.

However, the code for the blocks on its own is not sufficient to provide target ready code. For instance, it is still necessary to merge the blocks code into tasks and allocate those to cores etc. This is done by the operating system (OS) code generator. Among others, its purpose is to create the task code which will call blocks and distribute the code to the respective cores C files. Furthermore, it will create the input files for the targets compiler which will specify the mapping from tasks to cores as well as to provide the libraries for advanced controllers, e.g. optional CAN, LIN or Ethernet controllers.

7. CONCLUSION AND OUTLOOK

This paper introduces a hardware model which is capable of supporting automated partitioning and mapping in heterogeneous multi-core systems. The case study has shown how our hardware model is able to support the involved steps and which amount of hardware related information is required for an automated execution. Furthermore, it has outlined a feasible implementation of these aspects as part of a seamless tool chain.

Future work will target the development and implementation of advanced partitioning and mapping algorithms with different goals (i.e. energy efficiency), multiple constraints (e.g. bus access) as well as their support with hardware models.

8. REFERENCES

[1] P. Marwedel. *Embedded System Design: Embedded systems foundations of cyber-physical systems*, Springer Science+ Business Media, 2011.

[2] A. P. Consortium et al., EAST-ADL domain model specification, 2010. [Online]. Available: http://www.east-adl.info

[3] AUTOSAR, Automotive open system architecture, 2007. [Online]. Available: http://www.autosar.org

[4] D. Cordes, P. Marwedel, and A. Mallik. Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming, in *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2010, pp. 267–276.

[5] L. Krawczyk and E. Kamsties. Hardware models for automated partitioning and mapping in multi-core systems, in *Proceedings of the 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS 2013)*, Berlin, Germany, 12-14 September 2013, pp. 721-725.

[6] I. T. Foster. *Designing and Building Parallel Programs*. Reading, Mass.: Addison-Wesley, 1995.

[7] J. Herrington. *Code Generation in Action*, Manning Publications Co., USA, 2003.

[8] R. Hoettger, B. Igel and E. Kamsties. A novel partitioning and tracing approach for distributed systems based on vector clocks, in *Proceedings of the 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications* (IDAACS 2013), Berlin, Germany, 12-14 September 2013, pp. 670-675.

[9] M. Drozdowski. *Scheduling for Parallel Processing*, Springer London, 2010.

Lukas Krawczyk *received his B.Sc. degree in informatics from Dortmund University of Applied Sciences and Arts in 2009 and is currently working as research assistant within the AMALTHEA project. His scientific interests are: Embedded Systems Development, hardware level programing.*

Erik Kamsties *is a professor for software engineering and embedded systems at the Dortmund University of Applied Science and Arts. He received a Diploma (M.S.) from the Technical University of Berlin and a doctoral degree (Ph.D.) in computer science from the University of Kaiserslautern, (Germany).*

His current research focuses on requirements engineering, model-driven development, and embedded multi-core systems.

computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

SPOT PRICE PREDICTION FOR CLOUD COMPUTING USING NEURAL NETWORKS

Volodymyr Turchenko [1], Vladyslav Shultz [1], Iryna Turchenko [1], Richard M. Wallace [2], Mehdi Sheikhalishahi [3], Jose Luis Vazquez-Poletti [2], Lucio Grandinetti [3]

[1] Dept. of Info. Comp. Sys. & Control, Ternopil National Economic University, Ternopil, Ukraine,
{vtu, vshu, itu}@tneu.edu.ua
[2] Dept. of Computer Arch. & Automation, Complutense University, Madrid, Spain
wallacerim@aol.com, jlvazquez@fdi.ucm.es
[3] Dept. of Electronics, Comp. Sci. & Sys., University of Calabria, Rende (CS), Italy
alishahi@unical.it, lugran@unical.it

Abstract: Advances in service-oriented architectures, virtualization, high-speed networks, and cloud computing has resulted in attractive pay-as-you-go services. Job scheduling on such systems results in commodity bidding for computing time. Amazon institutionalizes this bidding for its Elastic Cloud Computing (EC2) environment. Similar bidding methods exist for other cloud-computing vendors as well as multi–cloud and cluster computing brokers such as SpotCloud. Commodity bidding for computing has resulted in complex spot price models that have ad-hoc strategies to provide demand for excess capacity. In this paper we will discuss vendors who provide spot pricing and bidding and present the predictive models for future short-term and middle-term spot price prediction based on neural networks giving users a high confidence on future prices aiding bidding on commodity computing.

Keywords: Spot Market; Cloud Computing; Resource Management; Neural Networks; Prediction.

1. INTRODUCTION

Cloud computing is seen as a hyper-specialization of general-purpose information technology with computing attributes and characteristics making it an ideal information technology delivery model. We assert that the real definition of cloud computing is the convergence of essential ideal characteristics of various distributed computing technologies. A cloud/grid computing system is reusing known economic models, but has wide variances due to large sets of variables for both the operation of the system and the hosting and execution of client applications. These variances provide new business models and markets that provide profitability to the system owner, but must also be offered at a unit price that is attractive to users so that the commodity system is sufficiently used to provide profit. This price-point is dependent on time of use, number of users, and capitalization cost. With ubiquitous cloud computing and grid computing technology, it can become economically beneficial as it is available at any time for everyone. Economic benefits of cloud and grid adoption are the main drivers as shown in the study by Armbrust [1]. Initially, cloud providers had only a fixed price for their service offerings [2-3]. As cloud systems grow larger and are partitioned into more unique configurations, this fixed price method becomes inefficient when total demand is much lower than data center capacity leading to under-use of the system. Cloud providers need an incentive mechanism to encourage users to submit more jobs. When total demand rises over data center capacity, it is desirable to provide an incentive to users to reduce their demand through raising per–unit costs, decreasing performance, or decreasing system availability.

We illustrate the spot price (user cost) and spot instance (system instance at that cost) mechanism on the example of the Amazon Elastic Computing 2 Service (EC2). In 2009, Amazon introduced a new set of spot instances to sell its unused data center capacity based on a new market mechanism offering a variable pricing method. With this service, users are able to bid for unused capacity. The spot price mechanism for EC2 shares many similarities with the standard uniform price auction mechanism. The

spot price charged for a request, may fluctuate depending on the supply of, and demand for, spot instance capacity. Spot prices are a tuple of {maximum price per hour the user wishes to pay for an instance type, the region desired, and the number of spot instances to run}. If the maximum price bid exceeds the current spot price, the job(s) will run until termination by the user or the spot price increases above the user set maximum price. The cost of spot instance hours are billed based on the spot price at the start of each hour an instance executes. If the user spot instance is interrupted in the middle of an hour of an instance use (because the spot price exceeded the user maximum bid price), the user is not billed for that partial hour of spot instance use. However, if the user terminates the spot instance a charge occurs for the partial hour of use.

Market driven resource allocation has been applied to grid computing environments [2-3]. Recently, it has also been adopted by cloud computing. The auction-based resource allocation mechanism in the cloud spot market causes the price of services to be dynamic. The auction-based mechanism tries to address the question of finding the best match for customer demanded services in terms of supply and price to maximize provider revenue and customer satisfaction. For the provider, we have revenue maximization, supply, and spot price; whereas for the customer, we have cost minimization, demand, and bid price.

Short term forecasting has been a key to economic optimization in the electric energy industry [4] and is essential for power systems planning and operation. An electricity costing model does not have a mechanism to store electricity as it can not store its service while a cloud system can, thus the floor of the electricity model can be much lower than that of a cloud system as electricity can not be stored in sufficient quantities to keep its floor higher. The alternative is to restrict generation and loose the currently produced power. In a cloud model, the system can be made idle, almost instantly, and await a price point when it would be profitable to operate. For both the cloud market and the electricity market, accurate forecasting is very important for both production and consumption of commodities like compute resources and electricity in order to optimize their buying and selling decisions.

In this paper, we demonstrate a neural network method to predict spot prices that can be useful to users of cloud computing for bidding on spot instances of cloud system providers.

2. RELATED WORK

2.1. SURVEY OF CLOUD AND GRID COMPUTING PROVIDERS

In an examination of the current literature available on more than 120 cloud IaaS and PaaS providers, over 98 percent of cloud and grid computing providers do not have a spot price and auction mechanism including Microsoft's Azure products and Google's Engine Products. The SpotCloud system is the only provider found that has a mechanism similar to the Amazon EC2, but it is devoid of many of the control, security, and ownership mechanisms that EC2 has. Also noteworthy is the OpenStack open source cloud infrastructure that many IaaS vendors are supporting as a competitor to Amazon EC2.

1) SpotCloud: SpotCloud is an IaaS cloud clearing-house from Enomaly Inc. SpotCloud brokers are buyers for, and sellers of, cloud infrastructure capacity. SpotCloud is a bidding exchange that establishes a "standard" computing unit across sellers for simplified user management allowing buyers to bid on a commodity product. Of note, the computing unit sold is "raw" as it does not offer service level agreements or value-added elements such as security or application restart and data backup.

Sellers of capacity join the exchange through the Enomaly Web site and installing its own cloud platform, Elastic Computing Platform (ECP), OpenStack, or other platforms. This is done via an API published by Enomaly. Sellers list the capacity, geography, and price requirements on the SpotCloud Web portal, where the information is presented to buyers. A blind-listing can be done if a seller feels that very low pricing on excess capacity may hurt its brand or impact direct sales channels otherwise the seller's name is listed.

Enomaly provides transaction monitoring, billing, buyer payment collection, and payment to sellers. Enomaly collects a percentage of the seller proceeds. In the SpotCloud model, much of the value that Enomaly brings to the complex cloud marketplace is convenience and simplification. Sellers have little administrative overhead, and their costs are aligned with revenue, since Enomaly's fee (a percentage of sales) covers marketing, sales, billing, collection, and other costs of doing business. Buyers are relieved of the burden of researching individual providers, and because the rates are posted, they can be assured that they are paying market-defined rates for the services they buy.

With only ten percent of U.S. businesses using IaaS, enterprises are cautious about entering the cloud due to concern over loss of control, security, performance of applications – factors that are not

addressed in a commodity cloud which may lead to simple bid, commodity cloud services being only useful for a few cases relegating an IaaS like SpotCloud a small market of wholesalers.

2) OpenStack: OpenStack is a classic "mash-up" of the right technology and user needs being in the right place at the right time. The OpenStack Foundation (http://www.openstack.org/foundation) has many organizations and companies contributing and using this cloud system software. The primary commercial lead in this effort is Rackspace. Currently VMware does not have a mechanism for spot pricing or an auction mechanism. Through the vCloud product, part of their software suite, VMware lists 177 companies that offer private and public IaaS cloud configurations. VMware will also support OpenStack by extending support for ESX hypervisor in OpenStack.

IBM is using OpenStack as a central part of its future cloud strategy in the IBM SmartCloud Orchestrator service. IBM developed the SmartCloud platform before OpenStack was founded in 2009-2010, but now is replacing that core component with OpenStack which will be the foundation of the company's cloud strategy. The SmartCloud Orchestrator service provides configuration of the compute, storage and networking resources needed applications run on the IBM SmartCloud platform. SmartCloud is a pay-as-you-go public cloud offering with components for private cloud or dedicated hosted infrastructure as an IaaS or PaaS. OpenStack is becoming a central cloud component for IBM, HP, Dell, Cisco, Red Hat, and Rackspace have also announced major initiatives around the OpenStack project.

While an interesting competitor to Amazon EC2, the OpenStack project is only an enabler of IaaS and PaaS sites and has no auction or spot price mechanisms.

2.2. SPOT MARKET PREDICTION IN THE CLOUD

Spot price and spot market prediction have a key role in the economics of the electric energy industry and is essential for power systems planning and operations as discussed in [4-5]. Also in the literature, there are neural network based techniques to forecast electricity spot price. In [4], neural network techniques based on short-term load forecasting is presented to predict short-term spot price in the Australian national electricity market.

In [6], characteristics of Amazon spot instances have been explored and the authors have done their comprehensive analysis based on one-year price history in four data centers of Amazon's EC2. They analyzed different types of spot instances that Amazon offers in terms of spot price and the inter-price time (time between price changes) and determined the time dynamics for spot prices by hour-in-day and day-of-week. Moreover, they have proposed a statistical model that fits well these two data series. The statistical models based on the mixture of Gaussian distribution with three or four components are able to capture spot price dynamics as well as the inter-price time of each spot instance. Their model exhibits a good degree of accuracy under realistic working conditions.

Amazon provides the price history to help customers decide their bids. Figure 1 shows an example of a price history graph obtained from [7]. Currently, Amazon EC2 spot instance services are available for eight types of virtual machines. Each virtual machine type has different resource capacities for CPU, memory and disk. Amazon EC2 runs one spot market for each virtual machine type in each geographical availability zone [8]. All spot markets share the free data center capacity. This capacity is the remaining resources after serving all the guaranteed (i.e., contracted) instances.

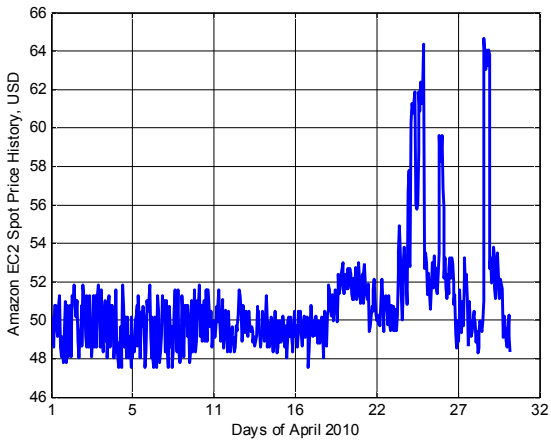


Fig. 1 – Amazon EC2 Spot Price History.

3. NEURAL-BASED PREDICTION METHOD

Over the past few decades, many different methodologies have been proposed for generating reliable predictions ranging from technical [9] and statistical analysis [10] to artificial intelligence techniques [11]. One of the artificial intelligence techniques, neural networks (NN), represent a promising alternative as the inherent learning ability allows effective capturing of the dynamic, nonlinear and complicated features of the predicted data. For example, a model of a feed-forward neural network, a multi-layer perceptron, showed excellent prediction results on many different financial examples [12–17]. Therefore, for the prediction of the spot prices we have used two standard models of

NNs: a multilayer perceptron (MLP), Fig. 2, and a recurrent neural network (RNN), Fig. 3 [11, 18]. These models are well researched and they are capable to fulfill approximation tasks with any required level of accuracy.

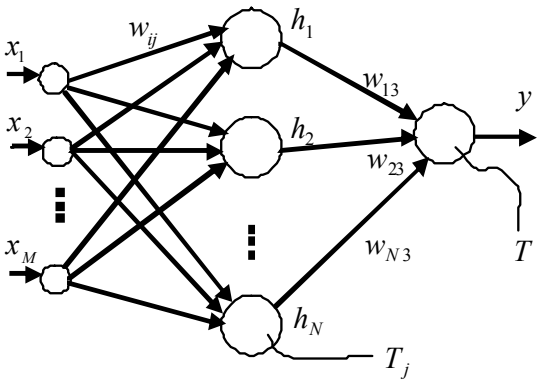


Fig. 2 – The structure of a three-layer MLP.

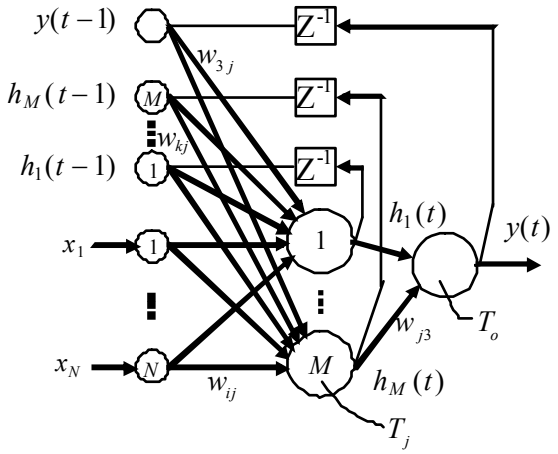


Fig. 3 – Structure of a recurrent neural network.

The output value of the three-layer perceptron can be formulated as:

$$y = F_3\left(\sum_{j=1}^{N} w_{j3}\left(F_2\left(\sum_{i=1}^{M} w_{ij}x_i - T_j\right)\right) - T\right), \quad (1)$$

where N is the number of neurons in the hidden layer, w_{j3} is the weight of the synapse from neuron j of the hidden layer to the output neuron, w_{ij} are the weights from the input neurons to neuron j in the hidden layer, x_i are the input values, T_j are the thresholds of the neurons of the hidden layer and T is the threshold of the output neuron [11, 18].

The output value of RNN can be formulated as:

$$y = F_3\left(\sum_{j=1}^{M} w_{j3}h_j - T_0\right), \quad (2)$$

$$y_j = F_2\left(\sum_{i=1}^{N} w_{ij}x_i + \sum_{k=1}^{M} w_{kj}h_k(t-1) + w_{3j}y(t-1) - T_j\right), \quad (3)$$

where M is the number of neurons in the hidden layer, w_{j3} is the weight of the synapse from neuron j of the hidden layer to the output neuron, N is the number of input neurons, w_{ij} are the weights from the input neurons to neuron j in the hidden layer, x_i are the input values, w_{kj} is the synapse from k context neuron of the hidden layer to j neuron of the same layer, $h_k(t-1)$ is the output value of k context neuron of hidden layer in the previous moment of time $t-1$, w_{3j} is the synapse from context output neuron to j neuron of the hidden layer, $y(t-1)$ is the value of context output neuron in the previous moment of time $t-1$, T_j are the thresholds of the neurons of the hidden layer and T_o is the threshold of the output neuron [11, 18]. The logistic activation function $F(x) = 1/(1+e^{-x})$ is used for the neurons of the hidden (F_2) and output layer (F_3) for the both MLP and RNN models. The standard back-propagation training algorithm [11] with constant or adaptive learning rate [20] is used for the training for both NN models.

Amazon EC2 provides spot instances from small standard systems to extra-large multiprocessor systems (at about 88 cores) and GPU co-processing. We have used historical data about spot prices of the "medium" cloud instances based on Linux and Windows operation systems called m1.linux and m1.windows respectively. These data are available on the Amazon web site [7]. For our experiments, we used 3842 spot price data points for 7 months starting from December 2009 and ending June 2010, which is a period of 215 days. This averages to 17 records of spot price for each day. We have divided all the data on appropriate months in order to do the experiments and visualization in a more efficient way.

For the input data analysis, it is beneficial to apply a moving simulation mode [14] since it provides the use of last recent data in the time series avoiding the impact of the "old" historical data on the quality of the prediction. The successful usage of the moving simulation mode for the financial application [19] showed that it is not necessary to choose a large data "window" for the analysis since a larger window will include the "old" historical data that makes the NN re-training less efficient.

Spot price prediction is beneficial in fulfilling short-term (single step) and middle- or long-term (multiple steps) predictions. The short-term prediction mode may provide better prediction

results since the preliminary analysis shows that the trend of data about spot price could change unpredictably fast and the short-term prediction could capture this change in an accurate manner. Since we have an average of 17 records about spot price per day, we have approximately 1.3 hours of time before new data arrives and, therefore, we can do the re-training of NN for each prediction step and improves prediction accuracy. On the other hand, the prediction interval of 1.3 hours is not suitable from the practical point of view since users of cloud resources want planning of their bidding strategies for several days ahead, with one to five days ahead being the most important.

4. EXPERIMENTAL RESULTS

We have formed the training set for our NN models using the Box-Jenkins method [10]. We have received the following results for the short-term [21] and middle-term prediction modes.

Short-Term Prediction Mode. Similar to [19], we have chosen 20 values as the size of the moving simulation window. The MLP architecture of 5-10-1 was chosen as the prediction structure in this mode. In particular, we chose 5 input neurons as it is sufficient within the 20 input data points of the moving simulation window with 10 neurons of the hidden layer being sufficient to provide a good generalization and prediction ability. We used the constant learning rate of 0.1 for both hidden and output layers of the MLP. The MLP is trained to reach the sum-squared training error of 10^{-5} with 2×10^{6} training epochs and then, on each step of the moving simulation mode, the MLP was re-trained using 7×10^{5} training epochs. One prediction step took about 12 seconds on a computer with an Intel Core 2 Duo processor at 2.4 GHz with 3 GB of RAM. The total computational time for the whole experiment in a short-term prediction mode was about 13 hours. According to the short-term prediction mode the real and predicted spot prices for both m1.linux and m1.windows cloud instances for each month from December 2009 through June 2010 are depicted in Figures 4 through 10. As shown in these figures, the MLP model in the proposed configuration provides a very good representation of the actual trending for both prediction cases. The numerical analysis of the predictions depicted in Table 2 shows the high accuracy of the proposed approach as the monthly average relative prediction errors do not exceed 5.6% for the m1.linux data and 6.4% for the m1.windows data. The average relative prediction errors for the whole testing period of six months are 3.3% and 3.7% respectively for m1.linux and m1.windows data. During the empirical analysis we have noticed that the amplitude of several data points is largely above or below of some average amplitude of signal change. In those points the prediction gives the result much different from this largely changed amplitude assuming that there should be the signal with much smaller amplitude. Therefore we have considered such prediction results as outliers. We have counted a prediction result as an outlier when its relative prediction error is more than 10%. The analysis of the prediction results shows that we have 155 (about 4.0% of the total results) and 188 (about 4.9% of the total results) outliers for the m1.linux and the m1.windows experiments respectively.

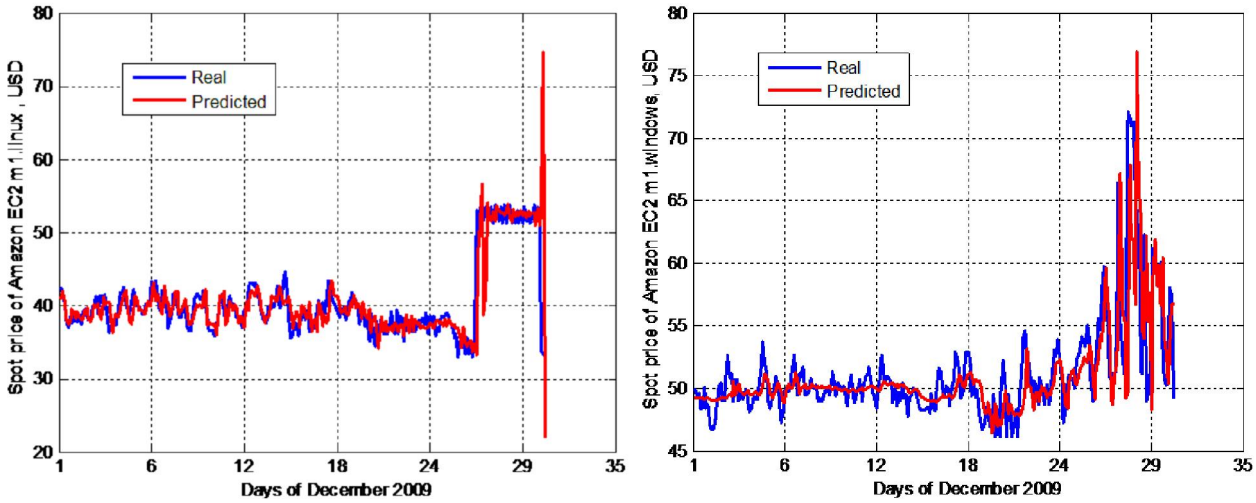


Fig. 4 – Short-term prediction results for m1.linux and m1.windows for Dec. 2009.

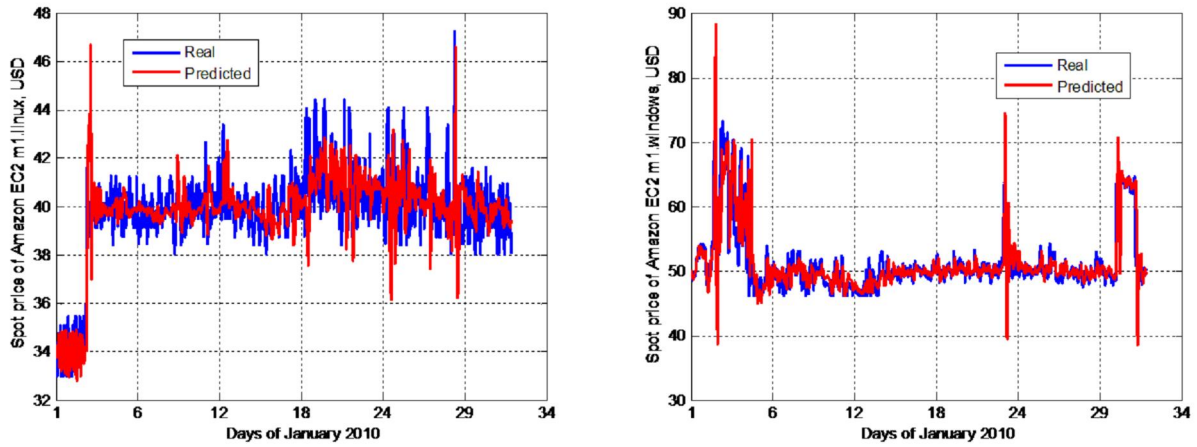


Fig. 5 – Short-term prediction results for m1.linux and m1.windows for Jan. 2010

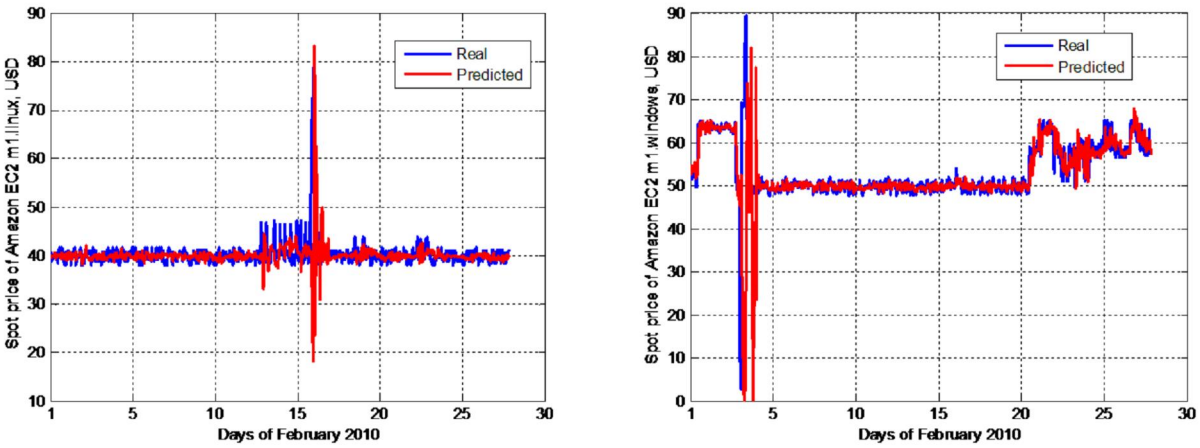


Fig. 6 –Short-term prediction results for m1.linux and m1.windows for Feb. 2010.

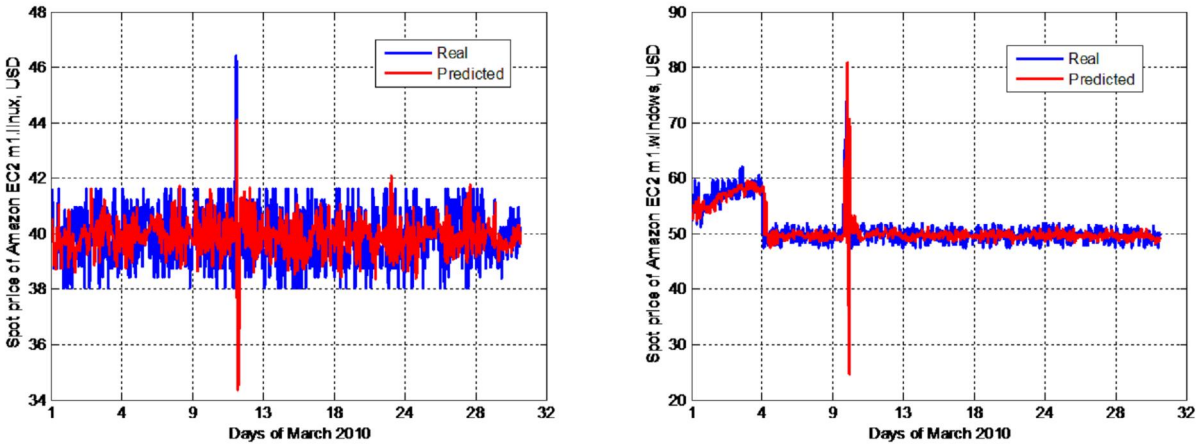


Fig. 7 – Short-term prediction results for m1.linux and m1.windows for Mar. 2010.

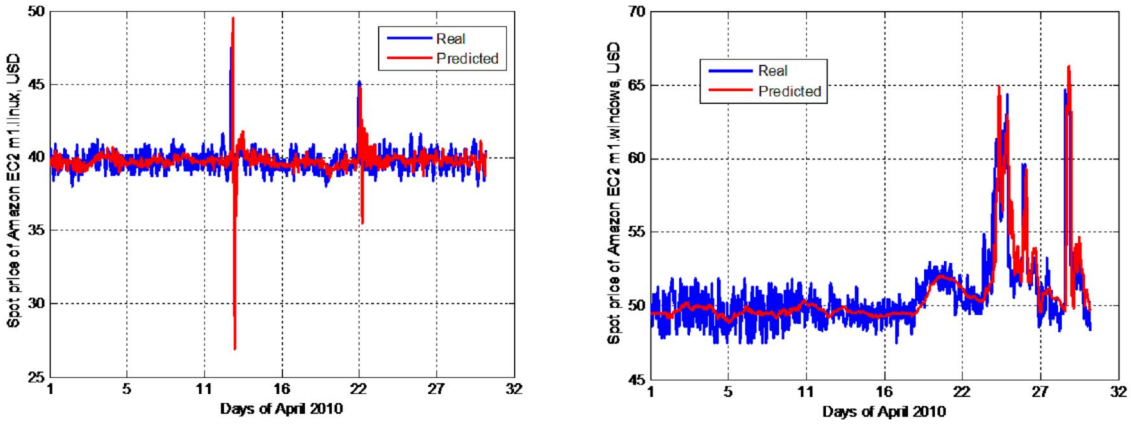


Fig. 8 – Short-term prediction results for m1.linux and m1.windows for Apr. 2010.

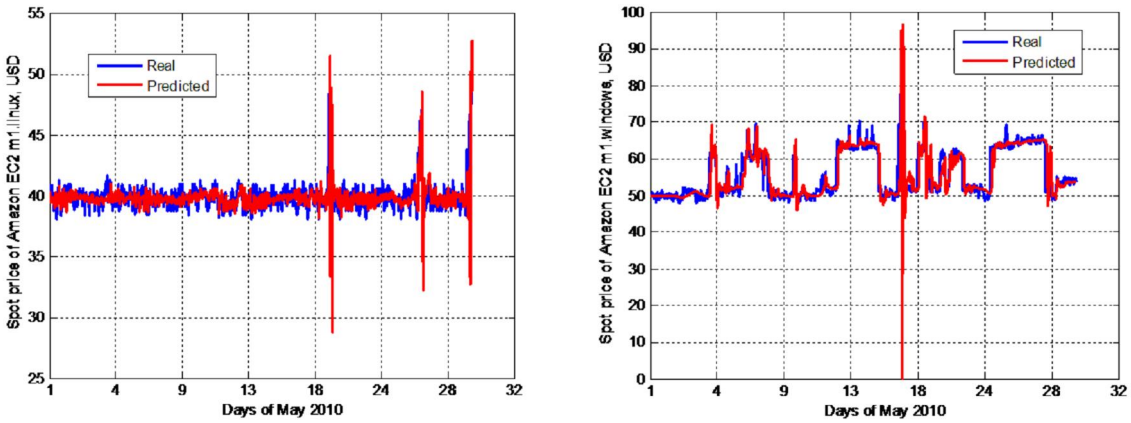


Fig. 9 – Short-term prediction results for m1.linux and m1.windows for May. 2010.

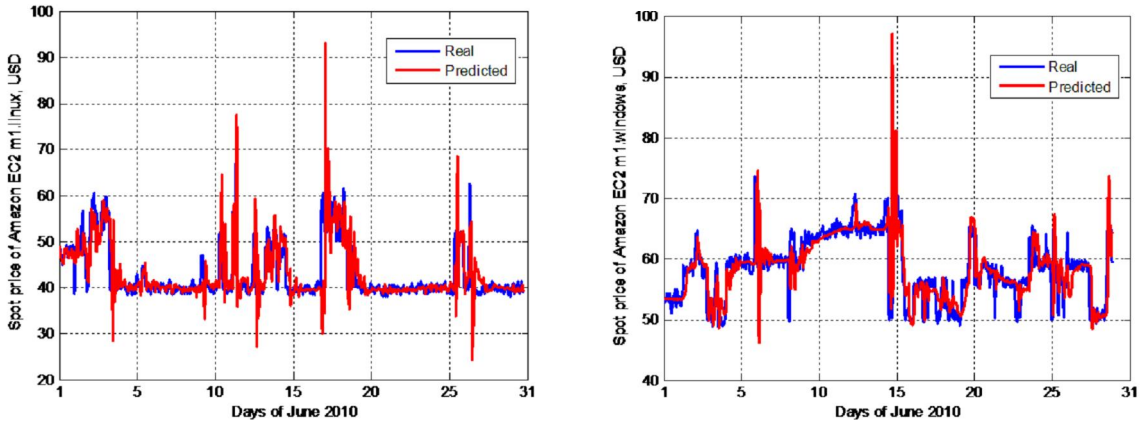


Fig. 10 – Short-term prediction results for m1.linux and m1.windows for Jun. 2010.

Table 1. Numerical results for short-term prediction mode.

Experiments	Avg. relative prediction error(%)		Num.& Percent of outliers (Rel. Predict. Err. >10%)	
	m1.linux	m1.windows	m1.linux	m1.windows
Dec.2009(266)	4.4	3.5	12 (4.5%)	16 (6.0%)
Jan.2010(556)	2.6	3.4	5 (0.9%)	25 (4.5%)
Feb.2010(556)	4.0	6.4	25 (4.5%)	28 (5.0%)
Mar.2010(663)	2.6	2.6	2 (0.3%)	10 (1.5%)
Apr.2010(564)	1.7	2.3	5 (0.9%)	7 (1.2%)
May 2010(637)	2.0	3.6	10 (1.6%)	49 (7.7%)
Jun.2010(595)	5.6	3.9	96 (16.1%)	53 (8.9%)
Average relative error/total (% number of) outliers:	3.3	3.7	155 (4.0%)	188 (4.9%)

Middle-Term Prediction Mode. Taking into account the long simulation time of the computational experiment above we have provided the middle-term prediction for the m1.linux data only. We have used 88 and 176 input data points from December 2009 to June 2010 as training data. We have used two NN models (MLP 5-10-1 and RNN 5-10-1) with reverse connections from both hidden and output layers. Both models use adaptive and constant learning rates. The constant learning rates were 0.5 and 0.5 for the hidden and output layers for the MLP model and 0.1 and 0.1 for the RNN model. Both models are trained to reach the sum-squared training error of 10^{-5} with 5×10^{5} training epochs. The training time of one middle-term prediction experiment took about 30 seconds using MLP model and 45 seconds using RNN model for the case of 88 input data points and about 60 seconds using MLP model and 90 seconds using RNN model for 176 input data points. All middle-term prediction experiments were executed on a Phenom II x 4 956 processor 3.4 GHz and 4 GB of RAM. The total computational time for the whole experiment in a middle-term prediction mode was about 4 hours. According to the middle-term prediction mode, the average and maximum relative prediction errors for one to five days for the four NN models are presented in Table 2 using training data for 88 data points; Table 3 presents data for 176 data points. The lower-case index values indicate the following: 1) the MLP model with adaptive learning rate; 2) the MLP model with constant learning rate; 3) the RNN model with adaptive learning rate; 4) the RNN model with constant learning rate. The graphical representations of middle-term prediction results for each testing month are detailed in Figures 11 to 14.

As can be seen the MLP and RNN models provide accurate prediction results for the majority of cases. For both of the 88 and 176 input training data sets the prediction results are a bit less accurate for the December 2009 and the June 2010 time periods on the fifth prediction day. Therefore the obtained results showed us good prediction abilities of neural networks for the middle-term prediction of spot prices of cloud resources.

Table 2. Numerical results for middle-term prediction using 88 training data points for each month.

Month	Relative prediction errors, %									
	1 day		2 days		3 days		4 days		5 days	
	avr	max	avr	max	avr	max	avr	max	avr	max
Dec 2009	4.3_2	8.4_2	4.0_2	11.4_2	4.5_2	11.5_2	4.1_2	11.5_2	4.3_2	14.7_2
Jan 2010	1.7_2	4.3_2	1.6_2	4.3_2	1.5_2	4.3_2	1.7_2	5.4_2	1.7_2	5.4_2
Feb 2010	2.0_4	4.3_4	2.4_4	4.9_2	2.5_2	4.9_2	2.4_4	5.0_4	2.4_2	5.0_4
Mar 2010	2.2_2	4.6_2	2.2_2	4.7_2	2.3_2	4.8_2	2.4_2	4.9_2	2.4_2	5.0_2
Apr 2010	1.2_1	2.2_1	1.2_1	2.2_1	1.3_2	2.8_2	1.3_2	2.9_2	1.4_2	3.4_2
May 2010	1.4_3	3.8_3	1.5_3	3.8_3	1.5_2	4.2_2	1.6_2	4.2_2	2.0_2	4.3_2
Jun 2010	2.4_3	8.2_3	2.6_3	11.5_3	2.8_3	11.5_3	3.1_3	11.5_3	3.4_3	11.5_3
Total average error	**2.2**	**5.1**	**2.2**	**6.1**	**2.4**	**6.3**	**2.4**	**6.5**	**2.5**	**7.1**

Table 3. Numerical results for middle-term prediction using 176 training data points for each month.

Month	Relative prediction errors, %									
	1 day		2 days		3 days		4 days		5 days	
	avr	max	avr	max	avr	max	avr	max	avr	max
Dec 2009	2.6_1	6.0_1	2.8_1	6.0_1	5.4_1	30.6_1	11.6_2	28.1_2	14.2_2	28.1_2
Jan 2010	2.0_1	4.5_1	2.1_1	5.7_1	2.1_1	5.8_1	2.0_1	5.9_1	1.9_1	6.0_1
Feb 2010	2.2_2	4.5_2	2.3_2	4.6_2	2.3_2	4.7_2	2.2_2	4.8_2	2.6_1	15.7_1
Mar 2010	2.1_2	4.1_2	2.1_1	5.0_1	2.2_1	5.1_1	2.3_1	5.2_1	2.5_2	13.7_2
Apr 2010	1.1_2	2.7_2	1.2_3	2.9_3	1.5_4	3.5_4	2.0_4	16.3_4	2.1_4	16.3_4
May 2010	1.3_2	2.5_2	1.4_2	3.5_2	1.4_2	3.8_2	1.4_2	3.8_2	1.5_2	4.4_2
Jun 2010	3.7_1	14.3_1	5.0_1	22.8_1	6.0_1	40.1_1	6.1_1	40.1_1	7.2_1	40.1_1
Total average error	**2.2**	**5.5**	**2.4**	**7.2**	**3.0**	**13.4**	**4.0**	**14.9**	**4.6**	**17.8**

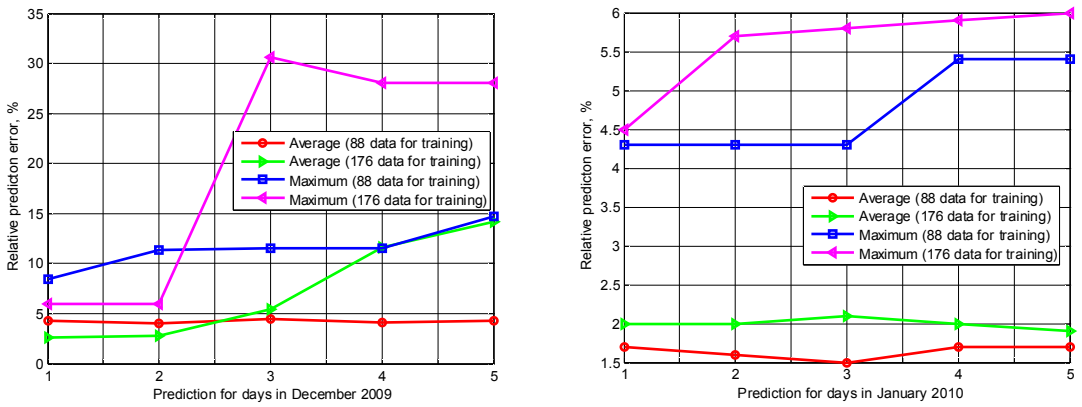


Fig. 11 – Middle-term prediction results for m1.linux for Dec. 2009 and Jan. 2010.

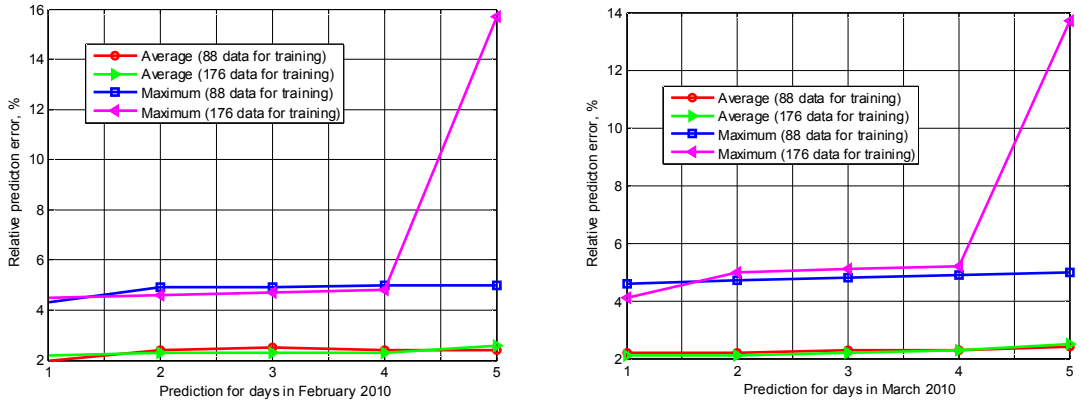


Fig. 12 – Middle-term prediction results for m1.linux for Feb. 2010 and Mar. 2010.

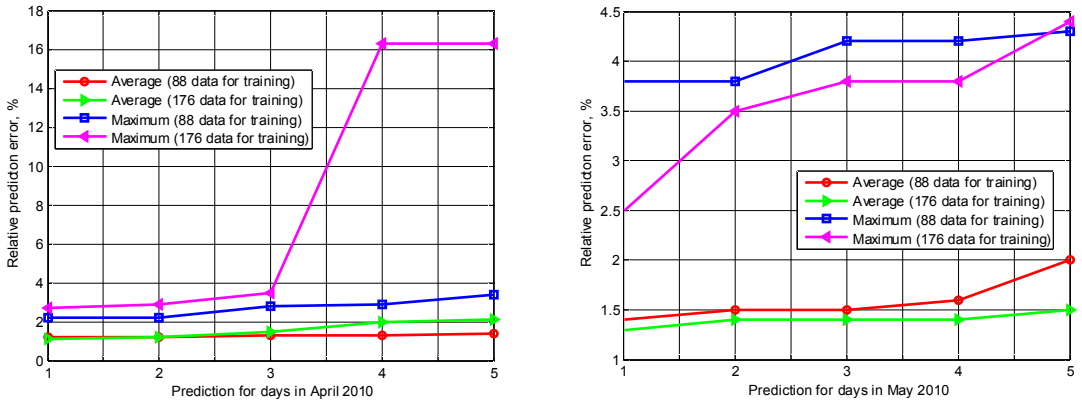


Fig. 13 – Middle-term prediction results for m1.linux for Apr. 2010 and May. 2010.

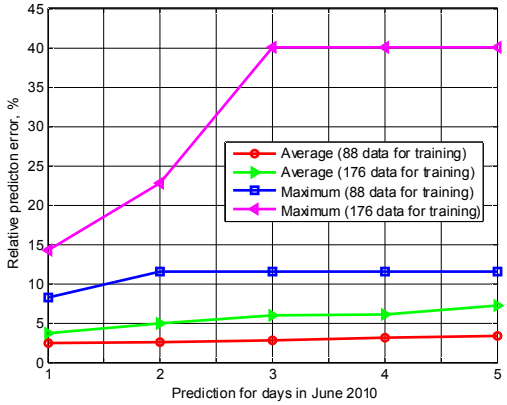


Fig. 14 – Middle-term prediction results for m1.linux for Jun. 2010.

5. CONCLUSIONS AND FUTURE WORK

Predictive models based on artificial neural networks for short-term and middle-term prediction of future spot prices for cloud computing are presented in this paper. Our models are based on standard multi-layer perceptron and recurrent neural network architectures. For prediction actions we used a moving simulation mode approach to remove old historical data for neural network re-training in order to improve a prediction accuracy of the model. The experimental results on the Amazon EC2 spot instances showed high prediction accuracy of the proposed approach. For the short-term prediction mode the average relative prediction error does not exceed 4% and the number of outliers (i.e., its relative prediction error is more than 10%) is not more than 5% for the total number of the prediction results. For the middle-term prediction mode, the average relative prediction error is in the range of 2.2 to 4.6% and the maximum relative prediction error is in the range of 5.1 to 17.8%. The obtained experimental results show that neural networks are well suited for such kind of prediction and could be very useful for users bidding on spot instance services.

Prediction of spot prices from other cloud service providers using neural networks will potentially be a future direction of our research.

REFERENCES

[1] Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia Above the clouds: A berkeley view of cloud computing, *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

[2] B. N. Chun, P. Buonadonna, A. Auyoung, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat, Mirage: A microeconomic resource allocation system for sensor net testbeds, in *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors*, 2005.

[3] C. Weng, M. Li, X. Lu, and Q. Deng, An economic-based resource management framework in the grid context, in *Proceedings of the IEEE International Symposium on Cluster Computing and the CCGrid'2005*, Vol. 1, 2005, pp. 542–549.

[4] P. Doulai and W. Cahill, Short-term price forecasting in electric energy market, in *Proceedings of the International Power Engineering Conference*, 17-19 May 2001, Singapore, Vol. 21, No. 2, 2001, pp. 29–29.

[5] T. Lora, J. C. R. Santos, J. R. Santos, J. L. M. Ramos, and A. G. Exposito, Electricity market price forecasting: Neural networks versus weighted-distance k nearest neighbours, in *Proceedings of the 13th International Conference on Database and Expert Systems Applications, ser. DEXA'02*, Springer-Verlag, London, UK, 2002, pp. 321–330.

[6] B. Javadi, R. K. Thulasiram, and R. Buyya, Statistical modeling of spot instance prices in public cloud environments, in *Proceedings of the UCC, IEEE Computer Society*, 2011, pp. 219–228.

[7] Amazon ec2 spot price history, 2012, [Online]. http://aws.typepad.com/aws/2011/07/ ec2-spot-pricing-now-specific-to-each-availability-zone.html

[8] Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, A view of cloud computing, *Communications of ACM*, Vol. 53, No. 4, April 2010, pp. 50–58. [Online]. http://doi.acm.org/10.1145/1721654.1721672

[9] J. J. Murphy, *Technical Analysis of the Financial Markets: a Comprehensive Guide to Trading Methods and Applications*, New York, NY: New York Institute of Finance, 1999.

[10] G. Box and G. Jenkins, *Time Series Analysis: Forecasting and Control*, Hoboken, NJ: John Wiley and Sons, 1970.

[11] S. Haykin, *A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1998.

[12] E. Guresen, G. Kayakutlu, and T. U. Daim, Using artificial neural network models in stock market index prediction, *Expert Systems with Applications*, Vol. 38, No. 8, August 2011, pp. 10 389–10 397. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2011.02.068

[13] Lin, Z. Yang, and Y. Song, Short-term stock price prediction based on echo state networks, *Expert Systems with Applications*, Vol. 36, No. 3, Part 2, 2009, pp. 7313–7317, [Online]. http://www.sciencedirect.com/science/article/pii/S0957417408006519

[14] R. R. Lawrence, *Using Neural Networks to Forecast Stock Market Prices*, University of Manitoba, 1997.

[15] S. H. Kim and S. H. Chun, Graded forecasting using an array of bipolar predictions: application of probabilistic neural networks to a stock market index, *International Journal of Forecasting*, Vol. 14, No. 3, 1998, pp. 323–337. [Online]. Available: http:

//www.sciencedirect.com/science/article/pii/S0169207098000003X

[16] X. Zhu, H. Wang, L. Xu, and H. Li, Predicting stock index increments b neural networks: The role of trading volume under different horizons, *Expert Systems with Applications*, Vol. 34, No. 4, 2008, pp. 3043–3054. [Online]. http://www.sciencedirect.com/science/article/pii/S0957417407002345

[17] Kara, M. Acar Boyacioglu, and O. K. Baykan, Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul stock exchange, *Expert Systems with Applications*, Vol. 38, No. 5, 2011, pp. 5311-5319. [Online]. http://dx.doi.org/10.1016/j.eswa.2010.10.027

[18] V. A. Golovko, *Neural networks: Training, models and applications*, Radiotechnika, 2011.

[19] V. Turchenko, P. Beraldi, F. De Simone, and L. Grandinetti, Short-term stock price prediction using mlp in moving simulation mode, in *Proceedings of the IEEE 6th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, Vol. 2, 2011, pp. 666–671.

[20] Golovko V., Savitsky J., Laopoulos Th., Sachenko A., and Grandinetti L. Technique of Learning Rate Estimation for Efficient Training of MLP, *IEEE-INNS-ENNS International Joint Conference on Neural Networks - IJCNN'00*, Como, Italy, July 2000, pp. 323-328.

[21] Wallace R.M., Turchenko V., Sheikhalishahi M., Turchenko I., Shults V., Vazquez-Poletti J.L., Grandinetti L. Applications of Neural-based Spot Market Prediction for Cloud Computing, in *Proceedings of the 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems IDAACS'2013*, Berlin, Germany, 12-14 September 2013, Vol. 2, pp. 710-716.

Dr. Volodymyr Turchenko *received his Diploma in System Engineering from Brest Polytechnic Institute, Brest, Belarus (1995) and his Ph.D. degree in Computer Engineering from National University "Lviv Polytechnics", Lviv, Ukraine (2001). He is now an Associate Professor of the Information Computing System and Control Department of Ternopil National Economic University (TNEU), Ternopil, Ukraine. He is a leader of Neural Network and Parallel Computing Research Group of the Research Institute for Intelligent Computer Systems in TNEU. He is a recipient of INTAS YSF Postdoctoral Fellowship (2004-2006) and FP7 Marie Curie IIF Postdoctoral Fellowship (2009-2012) both at the University of Calabria, Italy and Fulbright Visiting Scholar grant at the University of Tennessee, USA (2012-2013). He has published more than 100 publications. His main research interests are theory and application of neural networks and parallel computing.*

Mr. Vladyslav Shults *received his M.S. degree in Computer Engineering from TNEU, Ternopil, Ukraine (2012). He is now a junior researcher of the Information Computing System and Control Department of TNEU, Ternopil, Ukraine. He is a member of Neural Network and Parallel Computing Research Group of the Research Institute for Intelligent Computer Systems in TNEU.*

Dr. Iryna Turchenko *received her M.S. degree in Management Information Systems from Ternopil Academy of National Economy, Ternopil, Ukraine (1997) and her Ph.D. degree in Computer Engineering from TNEU, Ukraine (2008), where she is now an Assistant Professor. She has more than 30 international publications. Her main research interests are: theory and application of neural networks, multiparameter sensors and intelligent and web-based instrumentations.*

Mr. Richard Wallace *received his BS from the University of Idaho, Moscow, Idaho (1980); his Masters in Computer Science from the University of Dayton, Dayton Ohio (1986), and is currently pursuing his PhD in Computer Architecture from Complutense University of Madrid, Madrid, Spain under Dr. José Luis Vázque-Poletti. He is an Advising Engineer for unmanned system ground-stations and works with multiple aerospace companies in the Dayton, Ohio area. His work is focused on real-time, secure, highly available, distributed, life-critical, information critical, and security critical systems. His prior company, Quantum Solutions, developed several control system innovations in the telecommunications industry for LDAP and distributed systems for MCI, WorldCom, and British Telecom. In his early career, he led development of VHDL (IEEE-STD-1076) and lately actively contributed to PSL (IEEE-STD-1850). He has been the author of over 30 technical reports and*

publications. His main research interests are the theory and application of large distributed event-based systems for HPC and Cloud computing with an emphasis on hybrid computing architectures.

***Mehdi Sheikhalishahi** has 12 years of experience in Web technology, network security, and distributed computing technologies. Mehdi is currently a research fellow in Computer Science and Engineering department at University of Calabria in Italy after earning his PhD studies, and PhD thesis on energy efficient computing at the same department. In this position, he is doing research on cloud, green computing, and applications of cloud computing in scientific disciplines. Mehdi is co-author of several papers on grid, cloud computing, and energy efficiency. He was reviewer of several conferences and journals such as "Journal of Optimization Methods and Software", and "A Special Issue of IEEE Transactions on Parallel and Distributed Systems (TPDS) on Many Task Computing."*

***Dr. José Luis Vázquez-Poletti** received his M.E. in Computer Science from Universidad Pontificia de Comillas, Madrid, Spain (2004) and his Ph.D. in Computer Science from Universidad Complutense de Madrid, Madrid, Spain (2008), where he is now an Assistant Professor. He is a member of the Distributed Systems Architecture research group and has participated in 13 research projects funded by the European Commission and different Spanish ministries and agencies. He has more than 50 international publications. His main research interest is cloud computing and in particular, its application to high performance and high throughput applications.*

***Prof. Lucio Grandinetti** graduated from the University of Pisa, Italy and the University of California at Berkeley. He has been a post-doc fellow at University of Southern California, Los Angeles and Research Fellow at the University of Dundee, Scotland. Now he is Professor Emeritus at the Department of Computer Engineering, Electronics, and Systems of the University of Calabria, Italy. He was a member of the IEEE Committee on Parallel Processing, and European Editor of the book series of MIT Press on Advanced Computational Methods and Engineering. He is author of many research papers in well-established international journals and Editor (co-Editor) of several books on algorithms, software, applications in Parallel Computing, HPC, Grids and Clouds. He has been recipient and scientific leader of many European-Commission-Funded projects since 1993 (e.g. Molecular Dynamics Simulations by MPP Systems, EUROMED, HPC Finance, WADI, BEINGRID).*

Kathryn Dempsey, Vladimir Ufimstev, Sanjukta Bhowmick, Hesham Ali. A Parallel Template for Implementing Filters for Biological Correlation Networks, *International Journal of Computing*, Vol. 12, Issue 4, 2013, pp. 285-297.

High throughput biological experiments are critical for their role in systems biology – the ability to survey the state of cellular mechanisms on the broad scale opens possibilities for the scientific researcher to understand how multiple components come together, and what goes wrong in disease states. However, the data returned from these experiments is massive and heterogeneous, and requires intuitive and clever computational algorithms for analysis. The correlation network model has been proposed as a tool for modeling and analysis of this high throughput data; structures within the model identified by graph theory have been found to represent key players in major cellular pathways. Previous work has found that network filtering using graph theoretic structural concepts can reduce noise and strengthen biological signals in these networks. However, the process of filtering biological network using such filters is computationally intensive and the filtered networks remain large. In this research, we develop a parallel template for these network filters to improve runtime, and use this high performance environment to show that parallelization does not affect network structure or biological function of that structure.

Julián Lamas-Rodríguez, Francisco Argüello, and Dora B. Heras. Multiresolution rendering Based on GPGPU Computing, *International Journal of Computing*, Vol. 12, Issue 4, 2013, pp. 298-307.

The problem of visualizing large volumetric datasets is appealing for computation on the GPU. Nevertheless, the design of GPU volume rendering solutions must deal with the limited available memory in a graphics card. In this work, we present a system for multiresolution volume rendering which preprocesses the dataset dividing it into bricks and generating a compressed version by applying different levels of compression based on wavelets. The compressed volume is then stored in the GPU memory. For the later visualization process by texture mapping each brick of the volume is decompressed and rendered with a different resolution level depending on its distance to the camera. This approach computes most of the tasks in the GPU, thus minimizing the data transfers among CPU and GPU. We obtain competitive results for volumes of size in the range between 64^3 and 256^3.

Oleksandr Sudakov, Andrii Salnikov, Ievgen Sliusar, Oleksandr Boretskyi. Tools for Biomedical Data Archiving in Ukrainian Grid Infrastructure, *International Journal of Computing*, Vol. 12, Issue 4, 2013, pp. 308-315.

Tools for archiving and extraction of data in Ukrainian National Grid for end-users' applications are proposed, implemented and deployed for practical applications in medical imaging, non-linear dynamics, and molecular biology. Proposed tools provide the facilities to utilize large distributed storage space in grid infrastructures for different practical tasks including desktop applications. Tools may be successfully used even when on client platforms it is impossible to setup grid middleware, use web browser interfaces or grid security infrastructure authentication. Tools consist of extensible client compatible with different software and hardware platforms; web service for data transfer; web service for transparent data replication on grid storage elements.

Tevaganthan Veluppillai, Brandon Ortiz, Robert E. Hiromoto. Big Data Transfer for Tablet-Class Machines, *International Journal of Computing*, Vol. 12, Issue 4, 2013, pp. 316-323.

Several well-known data transfer protocols are presented in a comparative study to address the issue of big data transfer for tablet-class machines. The data transfer protocols include standard Java and C++, and block-data transfers protocols that use both the Java New IO (NIO) and the Zerocopy libraries, and a block-data C++ transfer protocol. Several experiments are described and results compared against the standard Java IO and C++ (stream-based file transport protocols). The motivation for this study is the development of a client/server big data file transport protocol for tablet-class client machines that rely on the Java Remote Method Invocation (RMI) package for distributed computing.

Robert Hoettger, Burkhard Igel, Erik Kamsties. Vector Clock Tracing and Model Based Partitioning for Distributed Embedded Systems, *International Journal of Computing*, Vol. 12, Issue 4, 2013, pp. 324-332.

Tracking, partitioning and tracing in modern dynamic high performance computing systems are three of the most innovative and important development aspects for performance optimization purposes and state-of-

the-art advanced quality. This paper discusses these three aspects with respect to distributed systems and proposes new mechanisms for an advanced utilization of software in this domain.

We present a specific tracking mechanism via vector clocks for model and code partitioning purposes and the determination of causality relations. Further, a tracing approach for an effective analysis and thereby utilization of code and the corresponding architecture is introduced. The combination of both approaches leads to a high degree of parallelism and a fine-grained structure of execution units, that further traced, supports a precise analysis of synchronous and asynchronous system's behavior as well as an optimal load balancing. The mechanisms are introduced with respect to a model based control engineering tool and event diagrams.

Vladislav Falfushinsky, Olena Skarlat, Vadim Tulchinsky. Integration of Cloud Computing Platform to Grid Infrastructure, *International Journal of Computing*, Vol. 12, Issue 4, 2013, pp. 333-339.

Both grid and cloud are used to organize large scale calculations and data processing on remote computers. Grid which became a basic computing infrastructure for the Large Hadron Collider experiments provides unified technical solutions for sharing and merging distributed heterogeneous computing resources within big collaboration groups. Cloud became popular among data centers and computing service providers because of flexibility, manageability and efficient hardware utilization. Both share common ideology "computing as a service", so one can expect additional benefits from their integration. The paper describes our approach to the integration. We propose to use cloud within grid sites for acceleration of application deployment and easy support of multiple virtual organizations by grid sites. The cloud in grid approach has been implemented and tested in Ukrainian National Grid, a part of European Grid Infrastructure.

Lukas Krawczyk, Erik Kamsties. Hardware Models for Automated Partitioning and Mapping in Multi-Core Systems using Mathematical Algorithms, *International Journal of Computing*, Vol. 12, Issue 4, 2013, pp. 340-347.

Multi-core CPUs offer several major benefits in embedded systems. For instance, they usually provide better energy efficiency and more computing power compared to single-core CPUs. However, these benefits do not come for free: A program has to be divided into tasks, which can be executed in parallel on different cores. Partitioning of software and mapping on cores are nontrivial activities that require detailed knowledge about the underlying hardware platform, e.g., the number of cores, their speed, available memories, etc. Such information is typically stored in handbooks. If this information would be available in a machine readable model, we call it hardware model, the partitioning and mapping activities can be automated. In this paper, we propose a hardware model and illustrate it using an example of a Freescale multi-core CPU. We then discuss a small case study, which illustrates the use of the hardware model in partitioning, mapping and code generation.

Volodymyr Turchenko, Vladyslav Shultz, Iryna Turchenko, Richard M. Wallace, Mehdi Sheikhalishahi, Jose Luis Vazquez-Poletti, Lucio Grandinetti. Spot Price Prediction for Cloud Computing using Neural Networks, *International Journal of Computing*, Vol. 12, Issue 4, 2013, pp. 348-358.

Advances in service-oriented architectures, virtualization, high-speed networks, and cloud computing has resulted in attractive pay-as-you-go services. Job scheduling on such systems results in commodity bidding for computing time. Amazon institutionalizes this bidding for its Elastic Cloud Computing (EC2) environment. Similar bidding methods exist for other cloud-computing vendors as well as multi–cloud and cluster computing brokers such as SpotCloud. Commodity bidding for computing has resulted in complex spot price models that have ad-hoc strategies to provide demand for excess capacity. In this paper we will discuss vendors who provide spot pricing and bidding and present the predictive models for future short-term and middle-term spot price prediction based on neural networks giving users a high confidence on future prices aiding bidding on commodity computing.

Kathryn Dempsey, Vladimir Ufimstev, Sanjukta Bhowmick, Hesham Ali. Паралельний шаблон для реалізації фільтрів для біологічних кореляційних мереж, *Міжнародний журнал Комп'ютинг*, том. 12, випуск 4, 2013, с. 285-297.

Висока пропускна здатність біологічних експериментів є основою їх використання в системній біології – здатність вивчати стан клітинних механізмів на великих вибірках відкриває для науковця можливість зрозуміти, як взаємодіють клітини та їх комплекси та як виявити хворобливі стани. Тим не менше, дані, отримані під час експериментів, мають великий об'єм і є різнорідними, вони вимагають використання методів штучного інтелекту для їх аналізу. Була запропонована кореляційна мережева модель для моделювання та аналізу цих даних з високою пропускною здатністю. Компоненти моделі визначаються за допомогою теорії графів. Вони були адаптовані для представлення ключових особливостей кліткових шляхів. Попередні дослідження виявили, що така мережа може покращити відношення сигнал-шум при біологічних дослідженнях. Тим не менше, процес фільтрування даних біологічних досліджень за допомогою використаних в мережі фільтрів має велику обчислювальну складність і фільтровані дані мають великий об'єм. У цьому дослідженні ми розробляємо паралельний шаблон для покращення цих мережевих фільтрів з метою зменшення часу виконання та використовуємо це високопродуктивне середовище для того, щоб показати, що розпаралелення не впливає на структуру мережі або її функціонування з точки зору біології.

Julián Lamas-Rodríguez, Francisco Argüello, and Dora B. Heras. Багатомасштабований рендерінг базований на GPGPU обчисленнях, *Міжнародний журнал Комп'ютинг*, том. 12, випуск 4, 2013, с. 298-307.

Проблема візуалізації значних об'ємних наборів даних постає при обчисленнях на графічних процесорних пристроях GPU. Однак проектування GPU-базованих рішень з об'ємного рендерінгу (побудови та відображення графічного зображення тримірного об'єкту) зіштовхується обмеженим об'ємом пам'яті графічної карти. У цій роботі ми представляємо систему для багатопараметричного об'ємного рендерінгу, що обробляє набір даних шляхом його поділу на блоки та генерування стиснутої версії з застосуванням різних рівнів стиснення на основі вейвлетів. Потім цей стиснутий об'єм зберігається в пам'яті графічної карти. Для подальшого процесу візуалізації способом накладання текстури, кожен блок даних розпаковується і відображається з різним рівнем роздільної здатності залежно від його відстані до камери. Цей підхід дозволяє обробити більшість завдань на графічній карті, зводячи до мінімуму обмін даними між процесором і GPU. Ми отримали конкурентноспроможні результати для об'ємів даних від 64^3 до 256^3.

Олександр Судаков, Андрій Сальников, Євген Слюсар, Олександр Борецький. Інструменти для архівування біомедичних даних в українській грід-інфраструктурі, *Міжнародний журнал Комп'ютинг*, том. 12, випуск 4, 2013, с. 308-315.

В цій статті запропоновано інструменти для архівування та отримання даних кінцевими користувацькими програмами Української національної грід-системи. Ці інструменти реалізовані та впроваджені для практичних застосувань в області медичної візуалізації, нелінійної динаміки та молекулярної біології. Запропоновані інструменти надають можливість використання розподілених елементів зберігання даних великого об'єму в грід-інфраструктурі для різних практичних завдань, включаючи застосування на робочих станціях. Інструменти можуть бути успішно використані, навіть якщо на клієнтських платформах неможливо встановити серединне програмне забезпечення грід, використати веб-браузер або аутентифікацію в грід-інфраструктуру. Інструменти складаються з розширеного клієнта, сумісного з різними програмними та апаратними платформами; веб-сервісу для передачі даних; веб-сервісу для прозорої реплікації даних на грід-елементах зберігання даних.

Tevaganthan Veluppillai, Brandon Ortiz, Robert E. Hiromoto. Передача великих обсягів даних для планшетних комп'ютерів, *Міжнародний журнал Комп'ютинг*, том. 12, випуск 4, 2013, с. 316-323.

Декілька відомих протоколів передачі даних представлено в порівняльному дослідженні для того, щоб дослідити питання передачі великих обсягів даних для планшетних комп'ютерів. Протоколи передачі даних включають стандартні Java і C++ протоколи, а також протоколи передачі блоків-даних, які використовують як Java New IO (NIO) і бібліотеки Zerocopy, так і протоколи передачі блоків-даних C++. Описано декілька експериментів та їх результати в порівнянні зі стандартними Java IO і C++ процедурами (потокові файлові транспортні протоколи). Мотивацією для цього

дослідження є розробка клієнт/серверного протоколу передачі файлів з великим обсягом даних для клієнтських комп'ютерів планшетного класу, що базується на методі віддаленого доступу (RMI) пакету Java для розподілених обчислень.

Robert Hoettger, Burkhard Igel, Erik Kamsties. Трасування за допомогою векторного тактового генератора та розподіл на основі моделі для дистрибутивних вбудованих систем, *Міжнародний журнал Комп'ютинг*, том. 12, випуск 4, 2013, с. 324-332.

Відстеження, розподіл та трасування в сучасних динамічних високопродуктивних обчислювальних системах є трьома найбільшими інноваційними та важливими аспектами проектування для забезпечення оптимізації продуктивності та сучасної кращої якості таких високопродуктивних систем. Ця стаття обговорює ці три аспекти стосовно розподілених систем і пропонує нові механізми для кращого використання програмного забезпечення в цій галузі.

Ми представляємо конкретний механізм відстеження через векторні тактові генератори для розподілу моделі і коду та визначення причинно-наслідкових зв'язків. Далі використовується трасування для ефективного аналізу і, таким чином, використання коду та відповідної архітектури . Поєднання цих двох підходів забезпечує високий ступінь паралелізму, деталізацію структури виконуваних блоків, що далі будуть трасуватися, , підтримує точний аналіз поведінки синхронних та асинхронних систем, а також оптимальний розподіл навантаження. Механізми вводяться стосовно моделі, заснованої на інженерних інструментах управління та діаграмах подій.

Владислав Фальфушинський, Олена Скарлат, Вадим Тульчинський. Інтеграція платформи хмарних обчислень в грід інфраструктуру, *Міжнародний журнал Комп'ютинг*, том. 12, випуск 4, 2013, с. 333-339.

Як грід, так і хмарні обчислення використовуються для організації складних обчислень та обробки великого об'єму даних на віддалених комп'ютерах. Грід, що є основною обчислювальною інфраструктурою для експериментів на Великому андронному колайдері, надає уніфіковані технічні рішення для поділу та об'єднання розподілених гетерогенних комп'ютерних ресурсів між великими робочими групами. Хмарні обчислення набули популярності серед дата-центрів та провайдерів обчислювальних сервісів внаслідок гнучкості, керованості та ефективності використання апаратних ресурсів. Обидві технології реалізують ідею "обчислень як сервісу", тож можна очікувати додаткових переваг від їх інтеграції. В статті описаний наш підхід до такої інтеграції. Ми пропонуємо використовувати хмарні обчислення у середині грід-сайтів для пришвидшення розгортання обчислювальних задач та спрощення підтримки грід-сайтами великої кількості віртуальних організацій. Підхід "хмарні обчислення у грід" впроваджений та пройшов тестування в Українській національній грід-системі, яка є частиною європейської грід-інфраструктури.

Lukas Krawczyk, Erik Kamsties. Апаратні моделі для автоматизованого розподілу та відображення в багатоядерних системах з використанням математичних алгоритмів, *Міжнародний журнал Комп'ютинг*, том. 12, випуск 4, 2013, с. 340-347.

Багатоядерні процесори надають кілька основних переваг у вбудованих системах. Наприклад, вони забезпечують кращу енергоефективність та більшу обчислювальну потужність в порівнянні з одноядерними процесорами. Однак, ці переваги не даються безкоштовно: програма має бути розділена на завдання , що можуть бути виконані паралельно на різних ядрах. Розподіл програмного забезпечення та відображення його частин на ядрах є нетривіальним завданням, що вимагає детальних знань про базову апаратну платформу, зокрема, про кількість ядер, їх швидкодію, доступню пам'ять, тощо. Ця інформація зазвичай зберігається в довідниках. Якщо ця інформація буде доступна комп'ютерній моделі (назвемо це моделлю апаратних засобів), то розподіл задач та їх відображення на ядрах можуть бути автоматизовані. У цій статті ми пропонуємо модель апаратних засобів та ілюструємо її на прикладі багатоядерних процесорів Freescale. Використання моделі апаратних засобів при розподілі, відображенні та генерації коду проілюстроване на прикладі тестового дослідження.

Володимир Турченко, Владислав Шульц, Ірина Турченко, Richard M. Wallace, Mehdi Sheikhalishahi, Jose Luis Vazquez-Poletti, Lucio Grandinetti. Прогнозування ціни ресурсів хмарних обчислень з використанням нейронних мереж, *Міжнародний журнал Комп'ютинг*, том. 12, випуск 4, 2013, с. 348-358.

Прогрес в сервісно-орієнтованих архітектурах, віртуалізації, високошвидкісних мережах та хмарних обчисленнях призвів до появи привабливих оплатних сервісів. Планування обчислювальних задач в таких системах є результатом аукціонних торгів за ресурси обчислювального часу. Компанія Амазон встановила практику таких аукціонних торгів для їхнього сервісу, що називається Elastic Cloud Computing (EC2). Подібні методи аукціонних торгів існують в інших провайдерів хмарних обчислень, а також у брокерів хмарних та кластерних обчислень таких як SpotCloud. Аукціонні торги за обчислювальні ресурси призводять до створення складних моделей ціни ресурсу, що мають спеціальні стратегії для забезпечення потреб в надлишкових ресурсах. В цій статті ми здійснили огляд провайдерів, хто забезпечує формування ціни за аукціонними принципами, та представили прогнозуючі моделі для майбутнього короткострокового та середньострокового прогнозування ціни обчислювальних ресурсів за допомогою нейронних мереж. Ми забезпечили високу точність прогнозу майбутньої ціни для його використання в аукціонних торгах за обчислювальні ресурси.

Kathryn Dempsey, Vladimir Ufimstev, Sanjukta Bhowmick, Hesham Ali. Параллельный шаблон для реализации фильтров для биологических корреляционных сетей, *Международный журнал Компьютинг*, том. 12, выпуск 4, 2013, с. 285-297.

Высокая пропускная способность биологических экспериментов является основой их использования в системной биологии – способность изучать состояние клеточных механизмов на больших выборках открывает для ученого возможность понять, как взаимодействуют клетки и их комплексы и как выявить болезненные состояния. Тем не менее, данные, полученные в ходе экспериментов, имеют большой объем и являются разнородными, они требуют использования методов искусственного интеллекта для их анализа. Была предложена корреляционная сетевая модель для моделирования и анализа этих данных с высокой пропускной способностью. Компоненты модели определяются с помощью теории графов. Они были адаптированы для представления ключевых особенностей клеточных путей. Предыдущие исследования показали, что такая сеть может улучшить отношение сигнал-шум при биологических исследованиях. Тем не менее, процесс фильтрации данных биологических исследований с помощью использованных в сети фильтров имеет большую вычислительную сложность и фильтрованные данные имеют большой объем. В этом исследовании мы разрабатываем параллельный шаблон для улучшения этих сетевых фильтров с целью уменьшения времени выполнения и используем это высокопроизводительную среду для того, чтобы показать, что распараллеливание не влияет на структуру сети или ее функционирования с точки зрения биологии.

Julián Lamas-Rodríguez, Francisco Argüello, and Dora B. Heras. Многомасштабированный рендеринг базированный на GPGPU вычислениях, *Международный журнал Компьютинг*, том. 12, выпуск 4, 2013, с. 298-307.

Проблема визуализации значительных объемных наборов данных возникает при вычислениях на графических процессорных устройствах GPU. Однако проектирование GPU-базированных решений из объемного рендеринга (построения и отображения графического изображения трехмерного объекта) сталкивается ограниченным объемом памяти графической карты. В этой работе мы представляем систему для многопараметрического объемного рендеринга, обрабатывающий набор данных путем его разделения на блоки и генерирования сжатой версии с применением различных уровней сжатия на основе вейвлетов. Затем этот сжатый объем сохраняется в памяти графической карты. Для дальнейшего процесса визуализации способом наложения текстуры, каждый блок данных распаковывается и отображается с разным уровнем разрешения в зависимости от расстояния до камеры. Этот подход позволяет обработать большинство задач на графической карте, сводя к минимуму обмен данными между процессором и GPU. Мы получили конкурентоспособные результаты для объемов данных от 64^3 до 256^3.

Александр Судаков, Андрей Сальников, Евгений Слюсар, Александр Борецкий. Инструменты для архивирования биомедических данных в украинской грид-инфраструктуре, *Международный журнал Компьютинг*, том. 12, выпуск 4, 2013, с. 308-315.

В этой статье предложены инструменты для архивирования и получения данных конечными пользовательскими программами украинской национальной грид-системы. Эти инструменты реализованы и внедрены для практических применений в области медицинской визуализации, нелинейной динамики и молекулярной биологии. Предложенные инструменты предоставляют возможность использования распределенных элементов хранения данных большого объема в грид-инфраструктуре для различных практических задач, включая применение на рабочих станциях. Инструменты могут быть успешно использованы, даже если на клиентских платформах невозможно установить программное обеспечение среднего слоя грид, использовать веб-браузер или аутентификацию в грид-инфраструктуру. Инструменты состоят из расширенного клиента, совместимого с различными программными и аппаратными платформами; веб-сервиса для передачи данных; веб-сервиса для прозрачной репликации данных на грид-элементах хранения данных.

Tevaganthan Veluppillai, Brandon Ortiz, Robert E. Hiromoto. Передача больших объемов данных для планшетных компьютеров, *Международный журнал Компьютинг*, том. 12, выпуск 4, 2013, с. 316-323.

Несколько известных протоколов передачи данных представлены в сравнительном исследовании для того, чтобы исследовать вопрос передачи больших объемов данных для планшетных компьютеров. Протоколы передачи данных включают стандартные Java и C++ протоколы, а также протоколы передачи блоков – данных, которые используют как Java New IO(NIO) и библиотеки Zerocopy, так и протоколы передачи блоков – данных C++. Описаны несколько экспериментов и их результаты по сравнению со стандартными Java IO и C++ процедурами (потоковые файловые транспортные протоколы). Мотивацией для этого исследования является разработка клиент/серверного протокола передачи файлов с большим объемом данных для клиентских компьютеров планшетного класса, основанный на методе удаленного доступа (RMI) пакета Java для распределенных вычислений.

Robert Hoettger, Burkhard Igel, Erik Kamsties. Трассирование с помощью векторного тактового генератора и распределение на основе модели для дистрибутивных встроенных систем, *Международный журнал Компьютинг*, том. 12, выпуск 4, 2013, с. 324-332.

Отслеживание, распределение и трассировка в современных динамических высокопроизводительных вычислительных системах являются тремя крупнейшими инновационными и важными аспектами проектирования для обеспечения оптимизации производительности и современного лучшего качества таких высокопроизводительных систем. Эта статья обсуждает эти три аспекта относительно распределенных систем и предлагает новые механизмы для лучшего использования программного обеспечения в этой области.

Мы представляем конкретный механизм отслеживания через векторные тактовые генераторы для распределения модели и кода и определения причинно-следственных связей. Далее используется трассировка для эффективного анализа и, таким образом, использование кода и соответствующей архитектуры. Сочетание этих двух подходов обеспечивает высокую степень параллелизма, детализацию структуры выполняемых блоков, которые дальше будут трассироваться, поддерживает точный анализ поведения синхронных и асинхронных систем, а также оптимальное распределение нагрузки. Механизмы действуют в отношении модели, основанной на инженерных инструментах управления и диаграммах событий.

Владислав Фальфушинский, Елена Скарлат, Вадим Тульчинский. Интеграция платформы облачных вычислений в грид инфраструктуру, *Международный журнал Компьютинг*, том. 12, выпуск 4, 2013, с. 333-339.

Как грид, так и облачные вычисления используются для организации сложных вычислений и обработки большого объема данных на удаленных компьютерах. Грид, который является основной вычислительной инфраструктурой для экспериментов на Большом адронном коллайдере, предоставляет унифицированные технические решения для разделения и объединения распределенных гетерогенных компьютерных ресурсов между крупными рабочими группами. Облачные вычисления приобрели популярность среди дата-центров и провайдеров вычислительных сервисов вследствие гибкости, управляемости и эффективности использования аппаратных ресурсов. Обе технологии реализуют идею “вычислений как сервиса”, поэтому можно ожидать дополнительных преимуществ от их интеграции. В статье описан наш подход к такой интеграции. Мы предлагаем использовать облачные вычисления в середине грид-сайтов для ускорения развертывания вычислительных задач и упрощения поддержки грид-сайтами большого количества виртуальных организаций. Подход “облачные вычисления в грид” внедрен и прошел тестирование в украинской национальной грид-системе, которая является частью европейской грид-инфраструктуры.

Lukas Krawczyk, Erik Kamsties. Аппаратные модели для автоматизированного распределения и отображения в многоядерных системах с использованием математических алгоритмов, *Международный журнал Компьютинг*, том. 12, выпуск 4, 2013, с. 340-347.

Многоядерные процессоры предоставляют несколько основных преимуществ во встроенных системах. Например, они обеспечивают лучшую энергоэффективность и большую вычислительную мощность по сравнению с одноядерными процессорами. Однако, эти преимущества не даются бесплатно: программа должна быть разделена на задачи, которые могут быть выполнены параллельно на разных ядрах. Распределение программного обеспечения и отражение его частей на ядрах является нетривиальной задачей, требующей детальных знаний о базовом аппаратную платформу, в частности, о количестве ядер, их быстродействии, доступной памяти и т.д. Эта информация обычно хранится в справочниках. Если эта информация будет доступна компьютерной модели (назовем это моделбю аппаратных средств), то распределение задач и их отражение на ядрах могут быть автоматизированы. В этой статье мы предлагаем модель аппаратных средств и иллюстрируем ее на примере многоядерных процессоров Freescale. Использование модели аппаратных средств при распределении, отражении и генерации кода проиллюстрировано на примере тестового исследования.

Volodymyr Turchenko, Vladyslav Shultz, Iryna Turchenko, Richard M. Wallace, Mehdi Sheikhalishahi, Jose Luis Vazquez-Poletti, Lucio Grandinetti. Прогнозирование цены ресурсов облачных вычислений с использованием нейронных сетей, *Международный журнал Компьютинг*, том. 12, выпуск 4, 2013, с. 348-358.

Прогресс в сервисно-ориентированных архитектурах, виртуализации, высокоскоростных сетях и облачных вычислениях привел к появлению привлекательных платных сервисов. Планирование вычислительных задач в таких системах является результатом аукционных торгов за ресурсы вычислительного времени. Компания Амазон установила практику таких аукционных торгов для их сервиса, под названием Elastic Cloud Computing (EC2). Подобные методы аукционных торгов существуют в других провайдеров облачных вычислений, а также у брокеров облачных и кластерных вычислений как SpotCloud. Аукционные торги за вычислительные ресурсы приводят к созданию сложных моделей цены ресурса, имеющие специальные стратегии для обеспечения потребностей в избыточных ресурсах. В этой статье мы осуществили обзор провайдеров, обеспечивающих формирование цены по аукционным принципам, и представили прогнозирующие модели для будущего краткосрочного и среднесрочного прогнозирования цены вычислительных ресурсов с помощью нейронных сетей. Мы обеспечили высокую точность прогноза будущей цены для его использования в аукционных торгах за вычислительные ресурсы.

Authors Guidelines

Authors are encouraged to submit high quality, original work in English only that has not appeared in, nor is under consideration by, other journals. Extended versions of papers that have previously published in conference proceedings may also be considered (this must be indicated at the time of submission). Authors must submit their manuscript as a .doc, .docx, .rtf or .pdf file by e-mail to:

International Journal of Computing
Research Institute for Intelligent Computer Systems
Ternopil National Economic University
3 Peremogy Square,
Ternopil 46020, Ukraine
Phone: +380 (352) 43-5050 ext. 12-234
Fax: +380 (352) 47-5053
E-mail: **computing@computingonline.net** or **computing.journal@gmail.com**

Submission of .pdf file is highly appreciated instead of the hard copy.

In case of any technical problems, contact us at **computing@computingonline.net** or **computing.journal@gmail.com**.

Journal Topics:

- Algorithms and Data Structure, Software Tools and Environments
- Bio-Informatics
- Computational Intelligence
- Computer Modeling and Simulation
- Cyber and Homeland Security
- Data Communications and Networking
- Data Mining, Knowledge Bases and Ontology
- Digital Signal Processing
- Distributed Systems and Remote Control
- Education in Computing
- Embedded Systems
- High Performance Computing and GRIDS
- Image Processing and Pattern Recognition
- Intelligent Robotics Systems
- Internet of Things
- IT Project Management
- Wireless Systems

Article Formatting

Essential title page information

• Title. Paper title has to be informative and not more than 50 characters. Please avoid abbreviations and formulae where possible.

• Authors' names and affiliations. Where the family name may be ambiguous (e.g., a double name), please indicate this clearly. Present authors' affiliation addresses (where the actual work was done) below the names. Indicate all affiliations with a lower-case superscript letter immediately after the author's name and in front of the appropriate address. Provide the full postal address of each affiliation, including the country name and, if available, the e-mail address of each author. Please designate a corresponding author.

Abstract

A concise and factual abstract is required. The abstract should state briefly the purpose of the research, the principal results and major conclusions. An abstract is often presented separately from the article, so it must be able to stand alone. For this reason, References should be avoided, but if essential, then cite

the author(s) and year(s). Also, non-standard or uncommon abbreviations should be avoided, but if essential they must be defined at their first mention in the abstract itself.

Keywords
Immediately after the abstract, provide a maximum of 6 keywords, using American spelling and avoiding general and plural terms and multiple concepts (avoid, for example, 'and', 'of'). Be sparing with abbreviations: only abbreviations firmly established in the field may be eligible. These keywords will be used for indexing purposes.

Introduction
There needs to be an adequate summary of references to describe the current state-of-the-art or a summary of the results.

Material and methods
Provide sufficient detail to allow the work to be reproduced. Methods already published should be indicated by a reference: only relevant modifications should be described.

Results
Results should be clear and concise.

Conclusions
The main conclusions of the study may be presented in a short Conclusions section, which may stand alone or form a subsection of a Discussion or Results and Discussion section.

Acknowledgements
Collate acknowledgements in a separate section at the end of the article before the references and do not, therefore, include them on the title page, as a footnote to the title or otherwise. List here those individuals who provided help during the research (e.g., providing language help, writing assistance or proof reading the article, etc.).

Subdivision
Divide your paper into clearly defined and numbered sections. Subsections should be numbered 1.1, 1.2, etc. (the abstract is not included in section numbering). Use this numbering also for internal cross-referencing: do not just refer to 'the text'. Any subsection may be given a brief heading. Each heading should appear on its own separate line.

Formulae
Mathematical expressions and Greek or other symbols must be written clearly with ample spacing.

Units
Follow internationally accepted rules and conventions: use the international system of units (SI). If other units are mentioned, please give their equivalent in SI.

Footnotes
Footnotes should be used sparingly. Number them consecutively throughout the article, using superscript Arabic numbers. Many wordprocessors build footnotes into the text, and this feature may be used. Should this not be the case, indicate the position of footnotes in the text and present the footnotes themselves separately at the end of the article. Do not include footnotes in the Reference list.

References
References must appear as a separate bibliography at the end of the paper, numbered by numerals in square brackets.

References to papers published in languages other than English must be translated into English indicating the original language given in brackets in the end of the reference e.g. (in German).

References in the text are indicated in square brackets. Journal titles must not be abbreviated. Please give journal volumes, issues (numbers) and page numbers.

Figures
Electronic version in EPS (Encapsulated Postscript) is preferred. The illustrations must be sharp, noise free and of good contrast.

Please do not:
• Supply files that are optimised for screen use (e.g., GIF, BMP, PICT, WPG); the resolution is too low;
• Supply files that are too low in resolution;
• Submit graphics that are disproportionately large for the content.

Plagiarism

The submissions will be checked on plagiarism and self-plagiarism on a regular basis, using the iThenticate plagiarism detection software, to ensure the originality of published material.

Each submitted paper is assigned to a three members of the Editorial Board for handling. The refereeing is done by anonymous reviewers.

When the paper is accepted, the authors will be required to electronically submit source files of the paper in MS Word. Each figure must also be supplied in two separate files, one in figure's original graphical format, and another in Postscript (.PS or .EPS) or .TIFF format.

The Technical Editor will send page proofs to the contact author for proofreading, usually by e-mail. The author is assumed to send the revised paper back within 10 days; otherwise the paper will be published without proofreading.

Copyright Notice

Privacy Statement

The names and email addresses entered into the International Journal of Computing site will be used exclusively for the stated purposes of this journal and will not be made available for any other purpose or to any other party.

TENTATIVE CALL FOR PAPERS

IDAACS'2015

The 8th IEEE International Conference on

Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications

September 24-26, 2015

WARSAW, POLAND

Organized by

Research Institute for Intelligent Computer Systems, Ternopil National Economic University and V.M. Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine

in cooperation with

Faculty of Electronics and Information Technologies, Faculty of Mathematics and Information Science, Warsaw University of Technology

Warsaw, Poland

www.idaacs.net

Conference Co-Chairmen

Anatoly Sachenko, Ukraine

Wiesław Winiecki, Poland

International Programme Committee Co-Chairmen

Robert E. Hiromoto, USA

Linas Svilainis, Lithuania

IDAACS International Advisory Board

Dominique Dallet, France

Richard Duro, Spain

Domenico Grimaldi, Italy

Vladimir Haasz, Czech Republic

Robert Hiromoto, USA

Theodore Laopoulos, Greece

George Markowsky, USA, Chair

Vladimir Oleshchuk, Norway

Fernando Lopez Pena, Spain

Peter Reusch, Germany

Anatoly Sachenko, Ukraine

Wiesław Winiecki, Poland

Important dates

Abstract submission: 1 February 2015

Notification of Acceptance: 31 March 2015

Camera ready paper: 15 May 2015

Early registration: 31 March – 15 June 2015

Correspondence

The correspondence should be directed to IDAACS Organizing Committee:

IDAACS Organizing Committee
Research Institute for Intelligent Computer Systems
Ternopil National Economic University
3 Peremoga Square
Ternopil 46020 Ukraine
Phone: +380352-475050 ext.: 12234
Fax: +380352-475053 (24 hours)
E-mail: orgcom@idaacs.net

Tentative Streams and Topics

The conference scope includes, but it's not limited to:

1. Special Stream in Wireless Systems
2. Special Stream in Project Management
3. Special Stream in Cyber Security
4. Special Stream in High Performance Computing
5. Special Stream in eLearning Management
6. Special Stream in Advanced Information Technologies in Ecology
7. Special Stream in Intelligent Robotics and Components
8. Advanced Instrumentation and Data Acquisition Systems
9. Advanced Mathematical Methods for Data Acquisition and Computing Systems
10. Artificial Intelligence and Neural Networks for Advanced Data Acquisition and Computing Systems
11. Bio-Informatics
12. Software Tools and Environments
13. Data Analysis and Modeling
14. Embedded Systems
15. Information Computing Systems for Education and Commercial Applications
16. Intelligent Distributed Systems and Remote Control
17. Intelligent Information Systems, Data Mining and Ontology
18. Intelligent Testing and Diagnostics of Computing Systems
19. Internet of Things
20. Pattern Recognition and Digital Image and Signal Processing
21. Virtual Instrumentation Systems

It's our pleasure to invite the interested scientists to organize own Streams.

Підписано до друку 27. 11. 2013 р.
Формат 60х84 $^{1}/_{8}$. Гарнітура Times.
Папір офсетний. Друк на дублікаторі.
Умов. друк. арк. 11,9. Облік.-вид. арк. 13,7.
Зам. № 004-13П. Наклад 300 прим.

Тернопільський національний економічний університет
вул. Львівська, 11, м. Тернопіль, 46004

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців ДК № 3467 від 23.04.2009 р.

Віддруковано у Видавничо-поліграфічному центрі «Економічна думка ТНЕУ»
46004 м. Тернопіль, вул. Львівська, 11
тел. (0352) 47-58-72
E-mail: edition@tneu.edu.ua