Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 100 (2015) 977 – 986

Procedia Engineering

www.elsevier.com/locate/procedia

25th DAAAM International Symposium on Intelligent Manufacturing and Automation, DAAAM 2014

A New Approach for Distributed Computing in Embedded Systems

Sergey Salibekyan[a]*, Peter Panfilov[a,b]

[a]*National Research University – Higher School of Economics, Myasnitskaya St. 20, Moscow 101000, Russian Federation*
[b]*Ministry of Economic Development, 1st Tverskaya-Yamskaya St. 1,3, Moscow 125993, Russian Federation*

Abstract

Historically, a typical embedded system has been designed as a control-dominated system using only a state-oriented model, such as FSMs. However, the trend in embedded systems design in recent years has been towards highly distributed architectures with support for concurrency, data and control flow, and scalable distributed computations. This implies that a different approach is necessary. We propose to use some dataflow computational model views to specify embedded systems, because it is a notation that covers the most relevant aspects of distributed computing. In this paper, we introduce a new computational model, known as OAA (Object-Attribute Architecture) and present the general characteristics of an OA-methodology to support the design and simulation of distributed computing systems. The matrix multiplication algorithm in the object-attribute distributed computing environment has been used to validate our methodology. The preliminary evaluation results show the feasibility of the OA approach.

Peer-review under responsibility of DAAAM International Vienna

Keywords: distributed systems; embedded computing; dataflow computational model; object-attribute architecture; computer system simulation

1. Introduction

An embedded computing and communications technology extends into all aspects of our everyday life from manufacturing and process automation, automotive and avionics control systems, through medical systems and personal health-care monitoring, local environmental monitoring for weather forecast and smart agriculture, to consumer electronics and home appliances, HVAC control and smart home automation systems. Within the more

* Corresponding author. Tel.: +7-495-916-89-09; fax: +7-495-917-81-54.
*E-mail address:*ssalibekyan@hse.ru

Peer-review under responsibility of DAAAM International Vienna
doi:10.1016/j.proeng.2015.01.457

general area of embedded computing the concept of distributed embedded systems refers to the collaboration among connected devices, such as tiny, stand-alone, embedded microcontrollers, networking devices, embedded PCs, robotics systems, computer peripherals, wireless data systems, sensors, and signal processors resulting in networked systems of embedded computing devices whose functional components are nearly invisible to end users [1].

There are number of key features that make distributed embedded systems different from traditional distributed systems. First is a tight coupling of the embedded system to the physical world. Typical embedded system is integrated into objects in the environment being it a factory floor or plant, vehicle, smart home or operating room at hospital, and is interconnecting hundreds, maybe thousands, of nodes. This will require changes in the way compute nodes interact with one another in a distributed computing environment. Second, components of the distributed embedded systems are typically small and highly resource constrained, wirelessly connected, bandwidth limited. Managing these constraints and creating a system that functions properly is a great challenge for distributed embedded system designers. Third, heterogeneity is the norm for distributed embedded systems because of the large number of interacting elements that make up system, so interoperability is a key concern. For the great number of connected applications running in diverse compute nodes it is no more feasible to use deterministic algorithms for resource and system management. Applications in this kind of system may merge and dissolve dynamically during system operation to host functionality which is specific to groups formed for that purpose. New approaches are required to naming, routing, security, privacy, resource management, and synchronization for successful implementation of such a dynamically reconfigurable distributed systems. Heterogeneity impact significantly the design process, implementation, and operation of distributed embedded systems, together with other concerns such as security, real-time programming, resource-bounded and energy-aware computing [2]. The distributed computer systems scenarios described later in this paper should clarify these points.

Modern intelligent manufacturing systems that include among others such an innovative concepts as bio-inspired self-organizing robotic systems [3] and bionic assembly systems (BAS) [4] require creating distributed embedded computing platforms that are themselves scalable, easily upgradeable and into which new technologies and new functions (applications) can be readily integrated. Traditionally, embedded systems have been designed as a control-dominated system using only a state-oriented model, such as FSMs. However, recent trend in embedded systems design towards highly distributed architectures with support for concurrency, data and control flow, and scalable distributed computations implies that a different approach is necessary. To answer these novel requirements, we propose to use some dataflow computational model views to specify embedded systems, because it is a notation that covers the most relevant aspects of distributed computing. In this paper, we introduce a new computational model, known as OAA (Object-Attribute Architecture) and present the general characteristics of an OA-methodology to support the design of distributed computing systems. Section 2 provides quick discussion on control flow and data flow computational paradigms in distributed systems domain. Section 3 describes the OAA. Section 4 presents the features of the OA-simulation environment in its application to the distributed system design. In Section 5, the matrix multiplication algorithm in the object-attribute distributed computing environment is used to validate our methodology. Initial evaluation results are shown in Section 6. Finally, some conclusions and future work are shown in Section 7.

2. Control flow vs. data flow in distributed computing

The emergence of Big Data, Cloud Computing, 4G Mobile data, Ubiquitous and Embedded Computing, and other modern trends drastically changed the spectrum of industry's tasks and problems and caused a prominent trend in distributed systems architecture design - a shift from powerful and expensive hardware towards a multitude of simple compute nodes (servers). These massively parallel (MPP) architectures can use either Google-like server farms built from off-the-shelf hardware or proprietary blocks forming what today is called a supercomputer. Contemporary parallel systems are rather often designed on the basis of distributed computer systems (CS) or server farms. This can be explained by the fact that the cost of a distributed CS consisting of serial computation modules united in a computer network is much lower than the cost of a single powerful supercomputer.

A factor restricting the development of such systems in distributed embedded systems domain is the control flow paradigm of computational processes which cannot be easily adapted to massively parallel and distributed computations and is thus a source of complexity in programming, synchronization, and computation scaling. The

application in distributed control flow systems consists of a set of isolated subsystems which operate at different compute nodes, exchange messages, and require additional synchronization mechanisms which impose additional requirements to programming skills of application developers. Problems may also arise when the target application (computational) problems are scaled as it is normally in case of distributed embedded systems application scenarios. It may happen that the application must be significantly modified (even completely rewritten) if the problem dimension and/or the underlying computer system configuration are changed dynamically. Current server farms or supercomputer architectures are not suited to support scalability at required level for applications with thousands or even millions of compute nodes. The MPI protocol (MPI library is the most popular tool for programming MPP-systems) is cumbersome – the procedure headers used to describe the data exchange contain a huge amount of parameters. Therefore, there is a clear need in design effective MPP-systems which are simple in programming, can easily be scaled, and are capable of self-parallel computations.

The use of dataflow systems, where computations are automatically synchronized according to the data circulating in a system [5], seems to be very promising in this respect. The dataflow concept also provides the solution to computation scaling problem, namely, the application code need not be rigidly assigned to certain computer system resources, which means that the resources are dynamically allocated during the application run. So far, the dataflow computational model has not been used widely in distributed systems domain, and not a single commercial general purpose dataflow computer system exists, although some dedicated FPGA-based dataflow systems are used rather often in R&D projects, and the programming languages constructed according to this principle become more and more popular (for example, Caltrop, Erlang [6]). Because of complexity of the general purpose dataflow CS architecture, the amount of required equipment is rather large, and this increases overall cost of the CS and affects its performance.

A new dataflow computational model called the object-attribute dataflow computer architecture (OAA) has been designed in the Moscow Institute of Electronics and Mathematics at the National Research University – Higher School of Economics (MIEM NRU HSE) [7, 8]. This architecture is rather simple, comparing to traditional dataflow models. The data communication token in OAA has the simplest format, and there is no need in complicated equipment for storage of the intermediate data required to seek the data package to perform the operation. We study the OAA efficiency for different classes of computational problems. Several methods for modeling the operation of computer system on the basis of Discrete Event Simulation (DES) model were developed to solve this problem [9]. This model can be used in modeling of different problems. In this paper, the main attention is paid to matrix multiplication simulation, because it is a primitive operation in the linear algebra which is widely used in scientific and industrial computing applications such as solution of SLAE and integration of systems of differential and ordinary equations, digital signal processing in embedded systems designs, etc. [10].

3. OAA architecture principles description

The OAA is extremely simple and consistent which ensures its very useful properties. One of such properties is the scalability which can be attained because OA-CS consists of three parts, namely: the hardware part (the OA-engine), the system part (which is called the OA-platform), and the virtual part (so called, OA-image).

The OA-engine is a distributed CS consisting of compute nodes (CN) (e.g., multiprocessor CS with common memory) united by communication links. The nodes and communication links can form a graph of any topology; the only requirement is that all of its nodes can communicate with each other (either directly or along a certain route passing through other nodes or functional units called router-schedulers), i.e., the graph must be connected. The OA-engine can be implemented on the basis of any computer hardware components including the standard ones (such as processors, memory, communication equipment) which allows for saving the overall CS cost.

The OA-platform is a set of algorithms for operation of functional units (FU) of all types present in OA-CS. An algorithm of FU function in dataflow computer architecture analyzes the obtained token with data and uses the token information field (attribute) to recognize the attached data and to decide how to process them. In the OAA, a token is called an information pair (IP) and the data processing algorithm depends on the IP attribute and on the FU state.

The OA-platform can be realized either by the hardware (if the FU operation algorithm is performed by some special equipment, as it is the case with embedded systems) or by the software (if the algorithm is realized as a

subroutine executed in the processor core).

The OA-image is a set of contexts (values of internal registers of all FUs participating in the computer process and a set of data to be processed). That means that the context is formed and initialized before the beginning of the process.

It is this division into three parts that ensures the OA-CS scalability property, because the OA-image describing the computational algorithm can be imposed on the distributed CS of any topology. The point is that the FUs form a unique address space in CS and this space is independent of the topology of the CS hardware part. This is realized as follows (see Fig. 1).

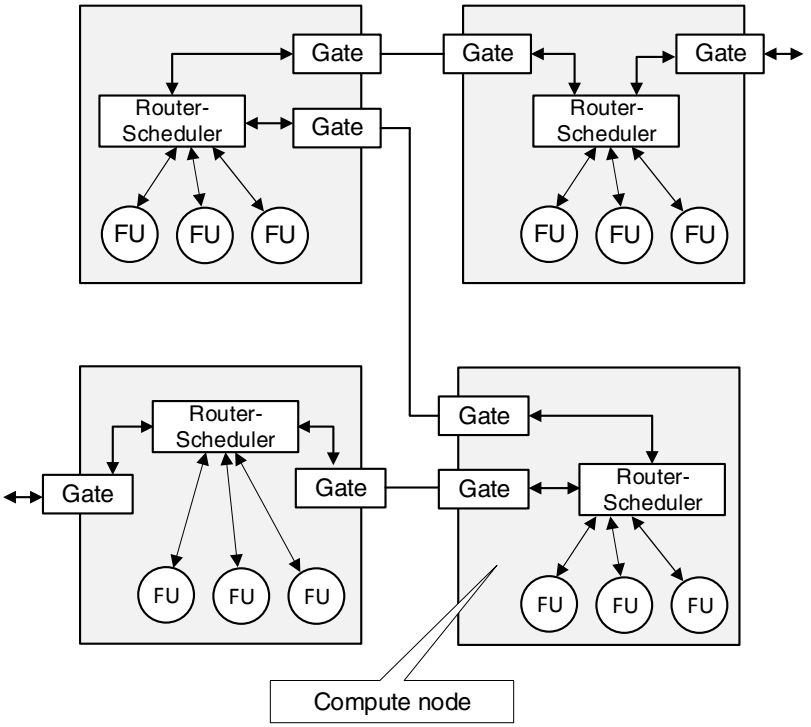


Fig. 1. Structure of distributed OA-CS.

The FU distributed over compute nodes exchange information in the IP form. The IP command field (attribute) permits uniquely determining FU or a group of FUs to which this IP is addressed. Therefore, each CN necessarily contains one or several Router-Scheduler FUs which ensure the IP delivery to the target addressee (addressees). The Router-Scheduler FU also allocates resources to other FUs (if IP arrives at a special FU, then some resources are allocated to it and the IP is given to it to be computed). If the addressee is at the same CN, then the IP is immediately given to it; otherwise, the Router-Scheduler FU transfers the IP through an information channel (gateway) to a different CN.

To launch the OA-image performance, it is necessary to distribute FUs over CNs and to adapt the Router-Scheduler FUs appropriately, so as they could determine the IP route to the addressee (addressees) and adapt the FU. If the OA-engine configuration is changed, then it is necessary to change the distribution of FUs over CNs and to readapt the Router-Scheduler FUs, but no change of the OA-image is required. This fact ensured the OA-CS scalability.

The Router-Scheduler FU adjusts the OA-CS hardware part to its OA-image part (Fig. 2). One of the functions of this FU is to allocate the hardware resources. This is necessary when IP arrives at FU to be processed. If all compute nodes (processor cores) are busy, then the request for allocation resources is placed in a queue to wait for the

resources and IP is placed in a queue to be processed by FU. The IP also gets in a queue to wait for the FU if the FU is processing the preceding IP. The OA-CS resources are allocated to FUs by the Router-Scheduler FU according to a certain algorithm (Fig. 2).

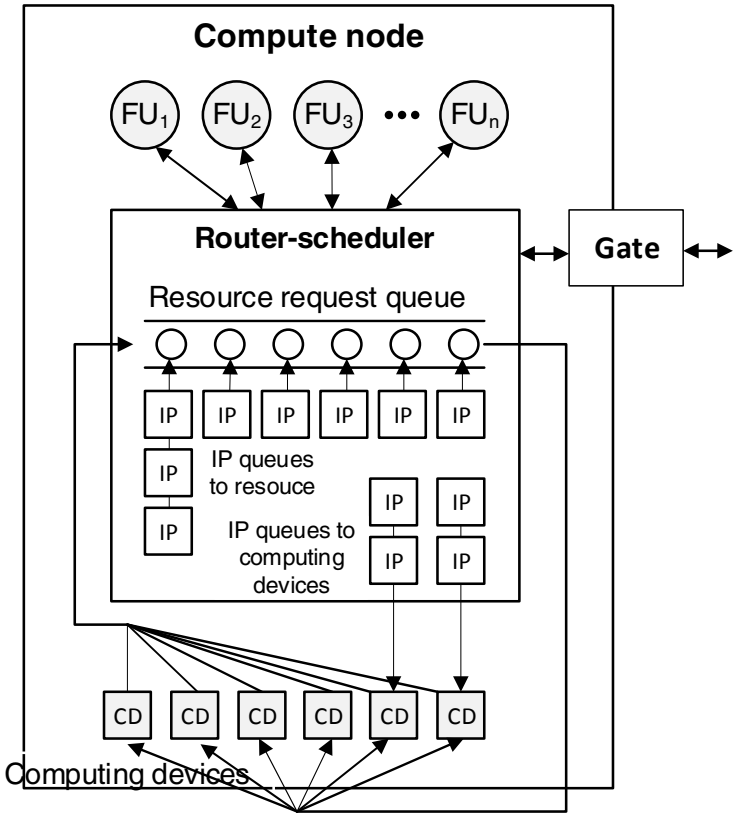


Fig. 2. Functional diagram of OA-CS compute node.

4. OAA simulation principles of the distributed computer system

The dedicated OA-programming and simulation environment was developed to study OA-CS capabilities and performance features. It permits simulation of computing processes in distributed OA-CS. This environment is a set of subprograms realizing the operational logic of the FUs composing an OA-platform. A specialized "dataflow-style" OA-language is used in OA-CS programming. This OA-language allows for specification both the OA-CS operation algorithm and its architecture. Also, an OA-language compiler is included in OA-programming and simulation environment, and facilities to introduce the simulation parameters for application debug and test. To perform the simulation, the OA-platform contains a dedicated Eventser FU to monitor the events occurring in simulation and a Scheduler FU emulating a Router-Scheduler FU in the CN. The simulation model can simultaneously host several Scheduler FUs (the Scheduler emulates operation of a separate CN) and only one Eventser FU because it monitors the sequence of events in the entire OA-CS.

In the OAA, a model of the computer system is constructed according to the discrete event principle (Discrete Events Simulation, DES [9]) where an event is the IP transfer from one FU to another FU. The simulation process is organized as follows (Fig. 3).

After IP arrives at a FU, the FU puts it in the waiting queue and sends a message to Scheduler FU about the requested operation duration (the access to Scheduler FU belongs to the context of the executive FU) and stays in

the sleep mode. The Scheduler FU allocates some resources to the FU and transfers the information about the FU forthcoming operation and its duration to the Eventser FU. The Eventser FU makes a mark on the time scale where it is necessary to perform the FU operation: $t = t_0 + \tau$, where t_0 is the current model time and τ is the proposed duration of the IP processing. On the Eventser FU's time scale, there may be several events at once (list of events), and Eventser FU activates the FU with the earliest activation mark, i.e., at the end of the list (EDF principle). The simulation process is complete when there are no marks in the list of events (Fig. 3) or, if necessary, another condition of the simulation process termination is introduced.

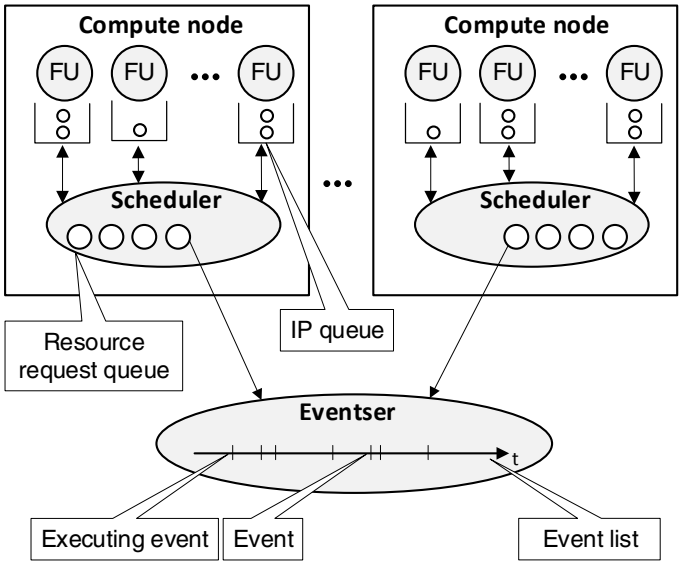


Fig. 3. Block-diagram of OA-CS simulation process.

5. Simulation example: matrix multiplication operation in object-attribute computer system

As an example of parallel processing application development and simulation using the OA-methodology, we chose the matrix multiplication application. The advantage of this application is that it can be used to verify the scalability of the distributed OA-CS and its operational efficiency. In literature, one can easily find parallel algorithms developed to accelerate the process of matrix multiplication [11]. Here we propose an approach that adopts the OAA model of dataflow parallel processing.

For the matrix multiplication application, a specialized OA-CS architecture is to be built according to the following principles. The entries of the initial matrices **A** and **B** (where **A** is an m-by-n matrix and **B** is an n-by-p matrix) are arranged into information pairs (IP). IPs with initial matrix entries are submitted to the executive FUs each responsible for computing a single entry of the resulting matrix **C**. Thus, the number of executive FUs in matrix multiplication OA-CS is equal to the number of entries of the resulting matrix **C**, where **C** is an m-by-p matrix whose entries are given by the dot product of the corresponding row of **A** and the corresponding column of **B**:

$$C_{i,j} = A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + \cdots + A_{i,n}B_{n,j} = \sum_{r=1}^{n} A_{i,r}B_{r,j},$$

where $1 \le i \le m$ and $1 \le j \le p$. The IP attribute obtained by the executive FU contains encoded indices of entries of the initial matrix and an indication of the initial matrix (A or B) it belongs to. The output of IP with the initial matrix data is performed by the Manager FU. The result obtained by the executive FU is also arranged as the IP and is submitted to the Collector FU to compose the resulting matrix.

We have studied two different topologies of FU interconnect in the OA-CS: the bus topology and the 2-dimensional grid (array) topology. In the bus topology, the model contains a dedicated FU emulating the common bus. The IPs entering the bus are submitted to the all FUs that examine the attribute and decide whether to read the data or not. The result of FU computations also enters the bus to be read by the Collector FU. To emulate SMP-systems, the executive FUs can be distributed over several CNs, i.e., they can be connected with different Scheduler FUs.

In the grid topology (Fig. 4), each executive FU is associated with a Router-Scheduler FU responsible for the data transmission (both the initial data delivery to FU and transmission of the output array elements calculated by the executive FUs to the Collector FU).

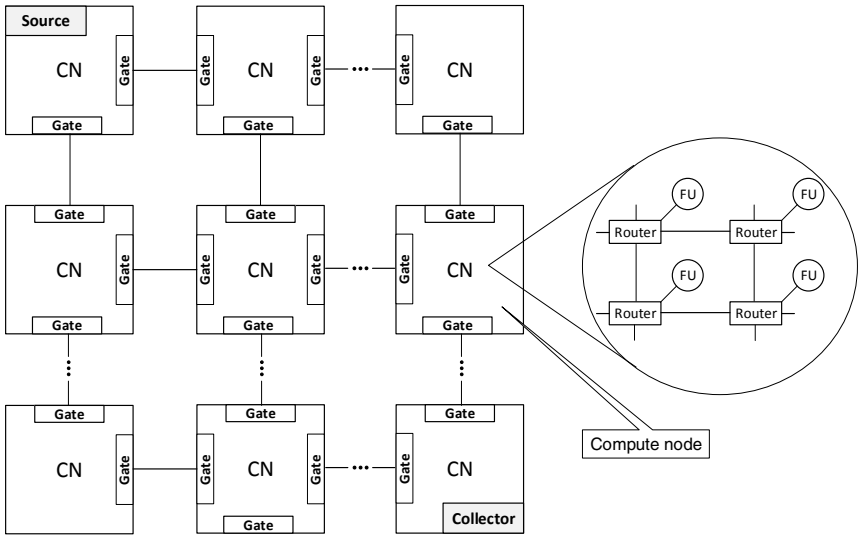


Fig. 4. Distributed OA-CS for Matrix Multiplication application of grid topology.

The simulation of the distributed CS was implemented in the grid topology. For this, a Gateway FU, i.e., a device emulating the data transmission through a communication link was introduced in the model. Each compute node containing a segment of the computational network was equipped with its own Scheduler FU emulating the distribution of resources in one CN.

The developed subroutines realizing the FU operational logic were plugged into the OA-programming and simulation environment, and the further study of the computational model was performed there. The environment incorporates an OA- programming language which allows for virtual FU creation, initialization, initial data input, obtaining the results of computations, and control the parameters of the simulated computational process.

The proposed simulation model allows us to monitor the number of parameters of the OA-CS operation including timing parameters of different FUs, network configuration parameters of the OA-CS in question, and resource usage parameters (memory capacity, buffer size) at compute nodes, etc..

6. Simulation results

The described approach was used to verify scalability of the OAA-based distributed computer system (OA-CS). This allowed us to determine the effect of the size of computational network (grid) on the computational time and the buffer capacity required to store the data at compute nodes and gateways. The buffer capacity was computed during the simulation model run as the sum of values of maximum lengths of the IP queues in schedulers and gateways at compute nodes (note: the model also allows us to determine the maximum lengths of IP queues for each segment of the computational network). This parameter is important because the buffers occupy a part of the main

memory and affects the overall cost of the computer system. The studies showed that in the case of a relatively small multiplication time with respect to the data transmission time, the efficiency of the computer system increases proportionally to the squared size of the grid (or, in other words, linearly in the number of compute nodes). This means that, for the n x n-grid, the efficiency is proportional to n^2. The capacity of buffers for the intermediate data storage increases proportionally to n^3, and the capacity of buffers for the data transmission increases proportionally to n^2. This result is independent on the computational segment size (we verified the *3*x*3, 4*x*4, 5x5*, and *10*x*10*-segments).

Figure 5 illustrates how the computation time and the capacity of the buffers in schedulers and gateways depend on the grid size for a system with the *2*x*2*-segment (the horizontal axis is the grid size). The graphs of the square root of the computation time, the cubic root of the scheduler buffer capacity, and the squared gateway buffer capacity are also given. These graphs are straight lines, and this fact confirms the polynomial dependence of parameters in question.

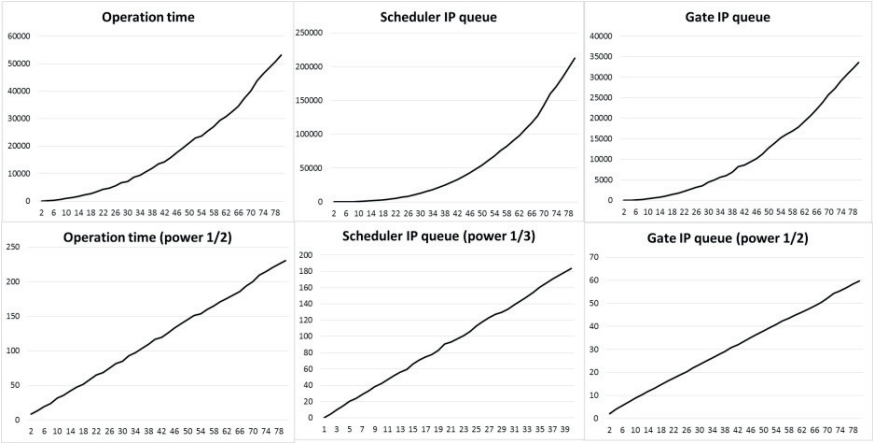
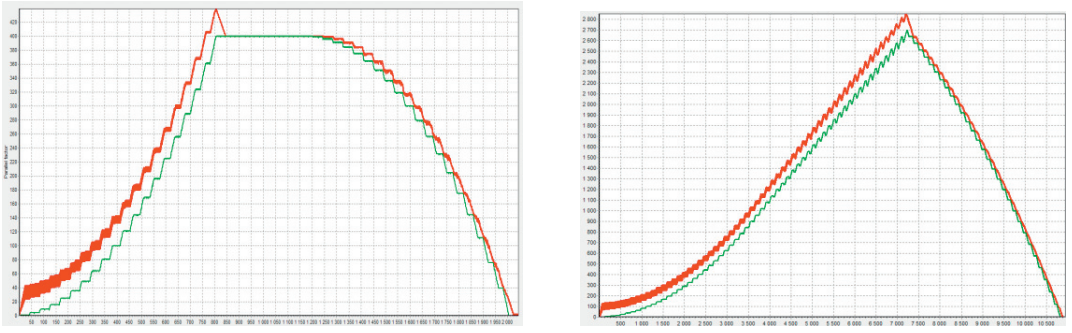


Fig. 5. Computational network parameters in function of the grid sizes.

If the multiplication time is greater than the time of data transmission through the FU network, then the computation time increases linearly, the scheduler buffer capacity increases quadratically, and the gateway buffer capacity increases linearly. This fact can be explained by the graph illustrating the computational process (Fig. 6). Here the green graph presents the number of FUs performing the operation of multiplication with accumulation, and the red graph presents the total number of operating FUs (executive FUs and Router-Scheduler FUs).

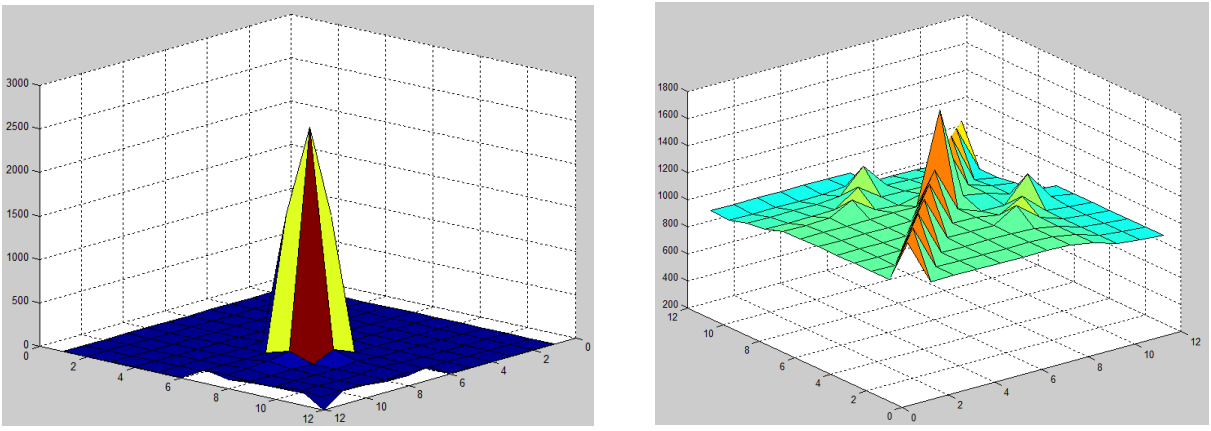


(a) (b)

Fig. 6. Computation parallelization results for (a) 15x15-Grid and (b) 60x60-Grid.

The first graph (a) illustrates the compute process where the time of the multiplication operation prevails. In this case, there is a time at which the initial data arrive at all executive FUs and there is a «plateau» area on the graph. Graph (b) illustrates computations with *60*-by-*60* matrices, and therefore the time of the initial data transmission over the network prevails, and the executive FUs begin to produce the results even before the time at which all initial data arrive at their destination. As a result, the compute process dynamics in case differs significantly.

The OA-simulation model of the distributed CS permits observing the distribution of required IP buffers over compute nodes. Fig. 7 illustrates the results of one execution of the model (number of required scheduler buffers and number of required gateway buffers per compute node). The simulation was performed for *60*x*60*-matrices with *5*x*5*-segments for different parameters of the model. The graphs illustrates the OA-CS operation in configuration with the source of initial data and the collector of the multiplication results (sink) both at the center of the compute grid.



(a) (b)

Fig. 7. Distribution of required IP buffers over compute nodes in a grid for (a) Schedulers and (b) Gateways.

The comparison analysis of different matrix multiplication methods adapted to the distributed computing environments [11, 12] shows that our method implemented according to the OAA dataflow computational model demonstrates some clear advantages over, for example, block methods (e.g., Fox's and Cannon's). The OAA-based method allows for quick start of computations well before the finish of input (matrix entries) data download, and for the early start of result (matrix C entries) upload – well before the end of computations. Block methods usually require that the information from entries in the initial matrices was distributed among computing blocks (CN) before the actual start of computation process, while the result upload only starts at the end of computations. A disadvantage of the OAA-based method is that it requires additional buffer memory for the intermediate data storage. Block methods are free from these circumstances.

Conclusion

An efficient and scalable distributed computing environments realization is among major foci of modern computer architecture R&D. In this paper, we propose the OAA (a dataflow computational model) to approach the problem of distributed embedded system design because it proved to be useful in the design of distributed computer systems and applications with the hardware part built around compute nodes of both traditional control flow (von Neumann) computer architecture and/or the newly redesigned dataflow computer architecture. An OA-approach to computer system design and simulation was demonstrated on the application problem of matrix multiplication as an example. The OAA-based matrix multiplication application was implemented. The distributed application was mapped onto the fine-grained reconfigurable distributed OA-CS and then analyzed using the special purpose

simulation model of the OAA. This OAA simulation environment is extensively used in the performance, efficiency, and scalability tests of distributed OA-CSs for the diverse classes of application problems (benchmarks). The OA-simulation model takes into account the basic parameters of the distributed computer system and allows CS architect to analyze the OA-CS architectural peculiarities, operational modes, and performance.

The simulation results have proved that the developed OA-CS solution for matrix multiplication problem can be implemented on both SMP and MPP systems and that the OA-CS is scalable, i.e., the CS can easily be adapted to any problem size, any network topology of compute nodes, and any number of executive FUs (processing elements) at a compute node. The performance of OA-CS in the case of small sizes of multiplied matrices is comparable with the performance of the block algorithm for matrix multiplication (which is proportional to N^3/P, where N is the dimension of a square matrix and P is the number of executive FUs [12]). In the case of large sizes of multiplied matrices, the computation time is dominated by the time of data transfer between CNs and is proportional to the number of CNs.

The results of this study were used not only for evaluation of the OAA solution to a specific application problem but also to work out the proposed OA-methodology for distributed dataflow embedded computer system design and simulation. Future plans of the research team include the extension of functionality of the OA-programming and simulation environment with the support of parallel/distribute DES model (PDES) [13]. It should permit to increase the simulation environment performance.

Acknowledgements

This work was partly funded by the project “Research of architectures of distributed dataflow computer systems for the semantic analysis of natural language”. The study was implemented in the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE) in 2014.

References

[1] Distributed Embedded Systems: Design, Middleware and Resources, B. Kleinjohann, L. Kleinjohann, W. Wolf (eds.), IFIP WCC 2008, Springer Science + Business Media, 2008 – 226 pages.

[2] H. Kopetz. Real-Time Systems: Design Principles for Distributed Embedded Applications. Second Edition. Springer Science + Business Media, 2011 – 378 pages.

[3] Kukushkin, I. K.; Katalinic, B.; Cesarec, P. & Kettler, R.: Reconfiguration in Self-Organizing Systems, Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, ISBN 978-3-901509-83-4, ISSN 1726-9679, pp 0641-0642, Editor B[ranko] Katalinic, Published by DAAAM International, Vienna, Austria 2011

[4] B. Katalinic, I. Kukushkin, V. Pryanichnikov, D. Haskovic. Cloud Communication Concept for Bionic Assembly System, Procedia Engineering, Volume 69 (2014), pp. 1562 – 1568. Available at: http://www.sciencedirect.com/science/article/pii/S1877705814004020.

[5] J. Silk, B. Robic and T. Ungerer. Asynchrony in parallel computing: From dataflow to multithreading. Institut Jozef Stefan, Technical Report CDS-97-4, September 1997.

[6] J. Armstrong, R. Virding, C. Wikström, M. Williams. Ericsson Concurrent Programming in ERLANG. Second Edition. Prentice Hall, 1996 – 351 pages. [PDF].

[7] S.M. Salibekyan, P.B. Panfilov. Object-attribute architecture for design and modeling of distributed automation system. Automation and remote control, Volume 73, 2012, Number 3, pp. 587-595. DOI: 10.1134/S0005117912030174

[8] P.B. Panfilov, S.M. Salibekyan. Dataflow Computing and its Impact on Automation Applications. Procedia Engineering, Volume 69 (2014), pp. 1286–1295. Available at: http://www.sciencedirect.com/science/article/pii/S1877705814003671.

[9] J. Banks, J.S. Carson II, B.L. Nelson, D.M. Nicol. Discrete-Event System Simulation. Fifth Edition. Prentice Hall, 2010 – 662 pages.

[10] Gr´egoire Allaire and Sidi Mahmoud Kaber. Numerical linear algebra, volume 55 of Texts in Applied Mathematics. Springer, New York, 2008. Translated from the 2002 French original by Karim Trabelsi.

[11] L.D. Jelfimova. A fast cellular method of matrix multiplication. Cybernetics and Systems Analysis, May 2008, Vol. 44, Issue 3, pp. 357-361.

[12] G. Nimako, E.J. Otoo, D. Ohene-Kwofie. Fast Parallel Algorithm for Blocked Dense Matrix Multiplication on Shared Memory Architectures. ICA3PP (1) 2012: pp. 443-457.

[13] R.M. Fujimoto. Parallel and Distributed Simulation; in Handbook on Simulation, J. Banks(ed.), Wiley, 1998.