computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

SYNCHRONIZATION BASED ON GLOBAL STATES AS A GENERAL CONTROL METHOD IN PARALLEL PROGRAMS

J. Borkowski[*], M. Tudruj[*][x], D. Kopanski[*]

[*]Polish-Japanese Institute of Information Technology,
86 Koszykowa Str., 02-008 Warsaw, Poland
[x]Institute of Computer Science, Polish Academy of Sciences
21 Ordona Str. 01-237 Warsaw, Poland
{janb, tudruj, damian}@pjwstk.edu.pl

Abstract: *New parallel program synchronization mechanisms are presented. A specialized synchronizer process, or a hierarchy of such processes, gather information about process states and construct Strongly Consistent Global States, using time interval timestamps. Global predicates evaluated by synchronizers can cause synchronization signals to be sent to processes, the signals trigger asynchronous computation activation or cancellation. The proposed framework is integrated with a message passing system - it is added to the GRADE graphical parallel programming environment to enhance its message-passing based features. Architecture and implementation of the enhancement are discussed.*

Keywords: *distributed systems, global states, global predicates, parallel programming, programming tools, visual programming.*

1. INTRODUCTION

Message passing has become one of the most popular and most successful parallel programming paradigms, especially due to standardization enabled by PVM and MPI libraries. Nevertheless, writing programs based on message passing libraries is still difficult since many technical details have to be known by a programmer. To ease parallel programming, program design tools are becoming popular. Message passing is lacking a systematic way for process synchronization. The code responsible for synchronization is mixed with the computational code and the synchronization conditions are expressed in terms of low-level operations (send/receive, barrier). In a program organized in a better way, data transfer operations should be separated from synchronization. Synchronization primitives should be transformed into a generalized system where synchronization is used as a top-level factor that constitutes a framework for general program execution control. Some proposals in this direction have been already published [TK98], however, they have not been implemented yet in practice. De-coupled implementation of synchronization in program execution control is consistent with the current tendencies of the systematic design of parallel systems. Much work has been done on efficiency of various forms of synchronization operations [CO94, OL95, SC96]. However, these tendencies are very scarcely supported by the design of adequate parallel program design tools. This project is an attempt to partially fill this gap.

An advanced synchronization environment for parallel applications has been proposed in [B00, B01]. To a big extent, the control in a parallel application program is dependent on synchronization, which is based on asynchronous evaluation of high-level conditions defined on application global states. The conditions are specified explicitly in special fragments of the program code. Processes react to a fulfilled synchronization condition in a way that is another novel feature. They can be temporally or permanently suspended if higher priority or more relevant actions are to be activated by a synchronization condition. In such situation, a process receives a synchronization message that immediately activates a procedure, which is an integral part of the application. In an alternative case, computations can be cancelled. Due to this some scarce computer resources can be liberated and made available for other tasks. In the proposed synchronization environment, synchronization-driven program execution control can be specified.

In this paper we describe how the ideas mentioned above can be combined with standard message passing in the graphical parallel program design environment GRADE [KDF97, KDFL99, PGR]. GRADE is meant for programmers who want to write parallel message passing programs without learning details of message passing library procedures. The programmer has only to draw a graph representation of a program divided into parallel processes, to fill the nodes with sequential C code and to assign which variables should be sent/received in which points of the program. GRADE presents a high level approach to parallel program design. However, it lacks more sophisticated and structured synchronization features that can be provided accordingly to above-mentioned principles. We discuss problems, which arise here and propose relevant solutions.

The paper is composed of 3 parts. In the first part, the idea of execution control in parallel programs based on process synchronization is explained. It includes verification of predicates defined on elements of global states in asynchronous systems and reactions on predicates as asynchronous activation and cancellation of computations in application programs. Next part contains a description of new synchronization features added to GRADE. In part 3, physical implementation issues of communication and synchronization for the proposed solution are discussed.

2. PROCESS SYNCHRONIZATION BASED ON GLOBAL PREDICATES

Global application states are used in parallel/distributed application monitoring and debugging to see if the application execution fulfils necessary conditions (global predicates) that warranties correct execution [CM91, GW94, M95, GW96, TG98, GM01]. We suggest using global predicates to control directly program behaviour. In this way, the control/synchronization scheme in a program can be made correct by construction and also immediately verifiable. Also, synchronization conditions expressed separately as global predicates will be easy to understand and modify.

The general idea of the proposed solution is shown in Fig. 1. Application program processes send messages on their states to special globally accessible processes called synchronizers. Each state message is labeled with a timestamp. A synchronizer collects state messages and determines if a consistent global state [CM91, G96, GW96, S00] has been reached. On each global state reached, one (or more) synchronization condition(s) (control predicate(s)) is (are) computed. If a predicate value is true, then a number of synchronization signals are sent by the synchronizer to selected application processes. On reception of these signals, application processes break their standard computation and the control in these processes is transferred to signal handling procedures. The procedures perform actions that constitute reactions to synchronization of application process states that have been reached.

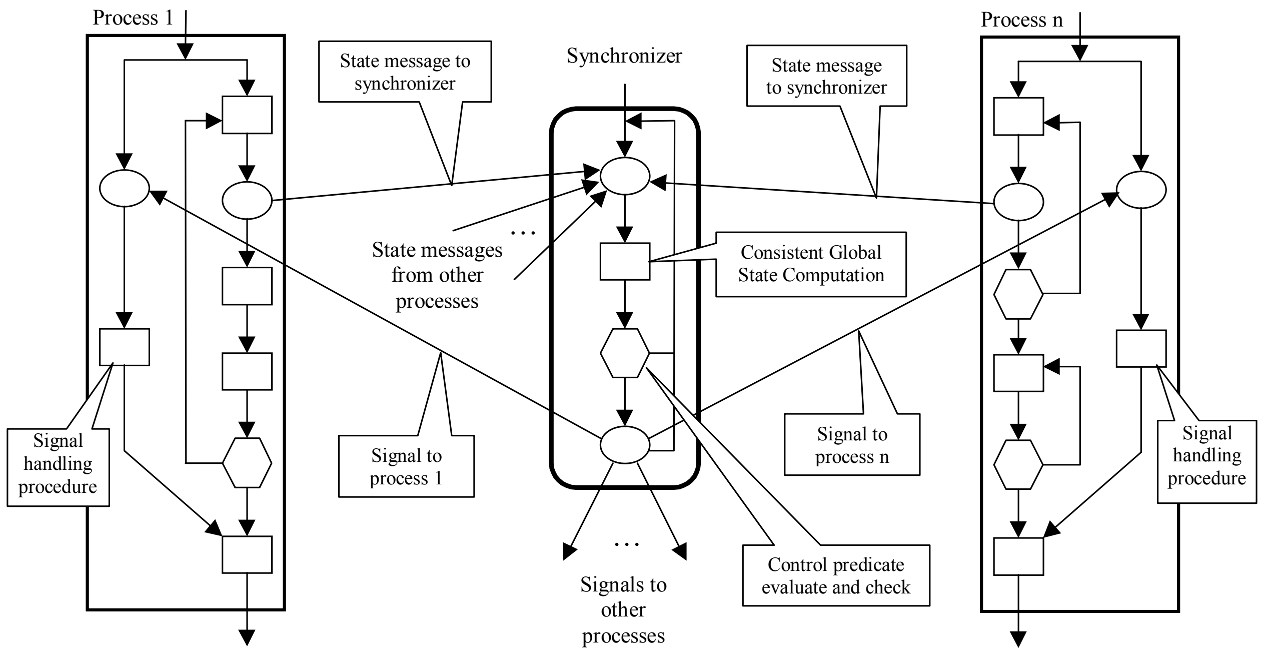
Process 1
State message to synchronizer
Synchronizer
State message to synchronizer
Process n
State messages from other processes
Consistent Global State Computation
Signal handling procedure
Signal to process 1
Signal to process n
Signal handling procedure
Signals to other processes
Control predicate evaluate and check

Fig. 1 – Computation of processes with a synchronizer

In a parallel system without common clock and without shared memory it is difficult to observe global states of applications. To solve the problem, logical vector clocks [M89, BM95] or partial synchronization of processor local clocks [S97, S00] can be used. Messages about process local states with attached timestamps should be sent to a synchronizer, and the synchronizer task is to combine the received information to identify consistent application global states. The actual sequence of global states cannot be observed with certainty; a synchronizer can only enumerate all the possible alternative execution scenarios. It is not possible to answer whether the actual application execution has passed through a state satisfying a given predicate, because we do not know which scenario happened in real and so, what are the exact states the application has passed through. This difficulty has led to a definition of global predicate modalities [CM91, GW94, FR95, GW96, S00]. Modalities give answers to questions concerning global predicate satisfaction. We need to know on-line, as early as possible, what is the actual application state. So we need a modality, which deals with real application states and with the actual application execution history, which can be evaluated on-line and which imposes low overhead. These conditions are met best by modality *Instantly* [S97, S00]. To be able to apply it, we need to synchronize process local clocks with an assumed tolerance ε. For a predicate φ, if *Instantly*(φ) is satisfied then there was a period in real time (and this period is known), when the application was in such a state, that φ was satisfied. Such states are called Strongly Consistent Global States (SCGS). It is possible for an application to pass a state satisfying φ, while *Instantly*(φ) is not detected, only when such a state lasts less then 2ε. Because this condition is clearly defined, a programmer can deal with it reasonably. The cost of SCGS detection is acceptable - O(E NlogN) [S97,S00], where E is the number of events at one process and N is the number of application processes. Other modalities have higher costs (even exponential) for unrestricted predicate forms. *Instantly* requires timestamps to be attached only to messages sent to a synchronizer, timestamps contain just two clock readouts.

Synchronizers observe application program states and evaluate pre-defined predicates. Whenever a predicate is satisfied, the synchronizer activates a reaction in some parts of executed application program. In a message passing system, it is done by sending to them control messages – signals. We want the processes to be able to react on signals possibly immediately by clearly defined actions. These goals are met by asynchronous activation and cancellation [B01, BKT03]. In the code of a process, designated regions are made sensitive to incoming signals. If the process control is inside a region sensitive to a signal of a given type a reaction is triggered when such a signal arrives. The reaction can be either activation or cancellation. Synchronization driven activation makes the current computation to be suspended and a reaction code associated with the region to be executed. After completion of the reaction code the suspended computing resumes. Synchronization driven cancellation makes the current computation to be stopped and a cancellation handling procedure associated with the region to be performed. The program execution resumes just after the abandoned region. Fig. 2. illustrates this concept.

An example of efficient program execution control of this type can be a branch-and-bound (B&B) algorithm [B00, BKT02]. The synchronizer knows the best solution found so far in a parallel B&B search. It can react immediately to prevent search processes from solving subtasks if their bounds are lower then the current best solution. A load balancing scheme can be included to the program implemented as an action activated by the synchronizer, triggered in global states with unbalanced load.

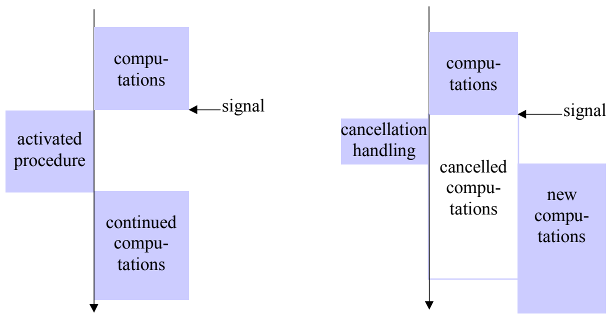

Fig. 2 – Principle of asynchronous activation (left part) and cancellation (right part)

3. IMPLEMENTATION OF PROPOSED SYNCHRONIZATION FEATURES IN GRADE

GRADE is a parallel programming environment based strictly on message passing. It can be extended by adding control and synchronization methods based on application global state analysis. GRADE allows a user to specify parallel processes, their interconnection and the internal structure of each process. The programmer specifies a program by the use of a graphical user interface and does not need to know any technical details of any communication library. Fig. 3 shows application processes and

communication channels with three processes connected by links through communication ports.
A separate window can be opened for each process, to specify the process behavior, drawn as a flow diagram, see Fig. 4. To create and modify such diagrams a programmer is supported with GRED graphical editor [KDFL99]. There are three main types of nodes used in the algorithm design: control statements (representing if, for, while,...), text blocks (sequential C language code can be put there) and communication nodes. Each communication node has communication ports assigned. In such a way the communication specified at the process level is translated onto the application level. The text blocks are filled with C code with the help of a text editor. The completed graphical program specification is translated into C language to be compiled and linked with GRADE libraries [DK99]

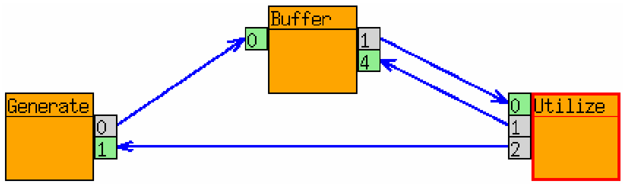

Fig. 3 – Processes and their interconnections in GRADE

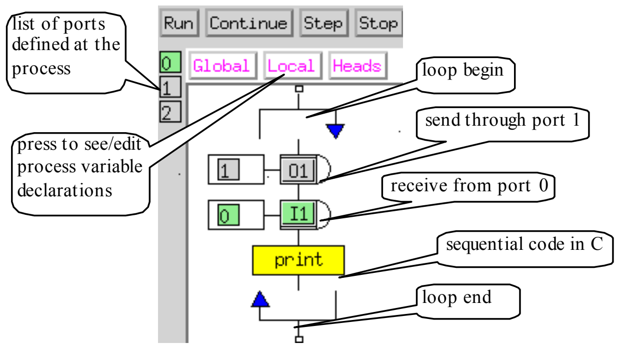

Fig. 4 – Flow diagram fragment of a process

A synchronizer, as described above, is represented as a special kind of a process. Using dedicated channels, application processes send to it messages about their local states, these messages are properly time stamped. The synchronizer constructs SCGSs using obtained information, evaluates defined predicates on them and sends back synchronization signals. We need here to specify how process state is expressed. The synchronizer has a number of input ports to receive state information from processes. The values sent to the ports are stored in arrays (after proper processing, see further explanations), one array per port, messages from process *i* are stored at array index *i*. So, the synchronizer sees the process states abstracted as values of array elements. Each array represents one aspect of process states, e.g. one can hold information about current workload, another about a problem currently being solved. Whenever a process wants to inform the synchronizer about a change in its local state, it sends a message with a proper value to a relevant synchronizer port. Synchronizer operates as shown in Fig. 5.

A synchronizer is shown at application level as a block similar to a standard computational process, Fig. 6. It has ports and channels that connect it with other processes. Its input ports accept process local state information messages, while output ports send synchronization signals. If we click on a synchronizer block, we open a separate window that shows its internal details, see Fig. 7. This window shows predicates as separate blocks. Attached input ports define the state information used by a predicate. A predicate evaluation can cause signals to be sent by attached output ports.

When we click on a predicate block, we get another window, in which we can specify the predicate details. The specification takes form of a control flow diagram. Execution of a predicate starts when the synchronizer reaches a SCGS. In the predicate control flow diagram, there are input statements which read in relevant values of a SCGS array bound to specified input ports. Predicates are calculated according to included definition. Output statements contained in the block dispatch synchronization signals. The signals are messages handled by processes in a special way.

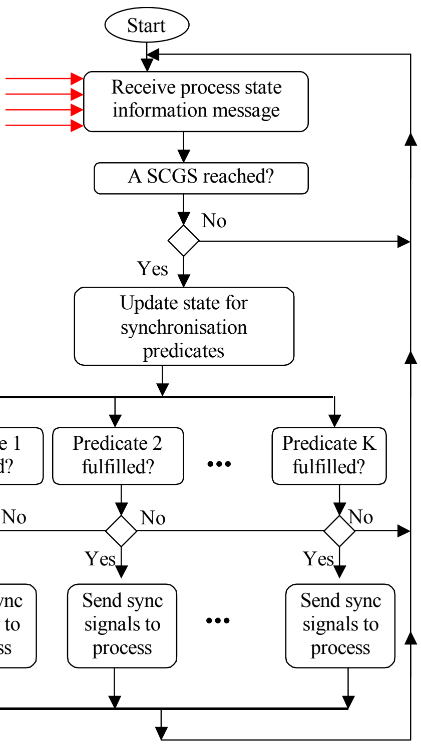
Start
Receive process state information message
A SCGS reached?
No
Yes
Update state for synchronisation predicates
Predicate 1 fulfilled?
Predicate 2 fulfilled?
...
Predicate K fulfilled?
No
No
No
Yes
Yes
Yes
Send sync signals to process
Send sync signals to process
...
Send sync signals to process

Fig. 5 – Control flow diagram of a synchronizer

Synchronization signals arriving at application processes can provoke a procedure activation or computation cancellation. GRADE process control flow diagram had been extended to express the new functionality. A simplified example of a control flow diagram made sensitive to synchronization signals is presented in Fig. 8. The normal execution flow goes along the path marked by a dotted line. If a signal arrives on ports 1 or 3 when the process execution is within a dashed rectangle, then the control is transferred to the right-hand side block.

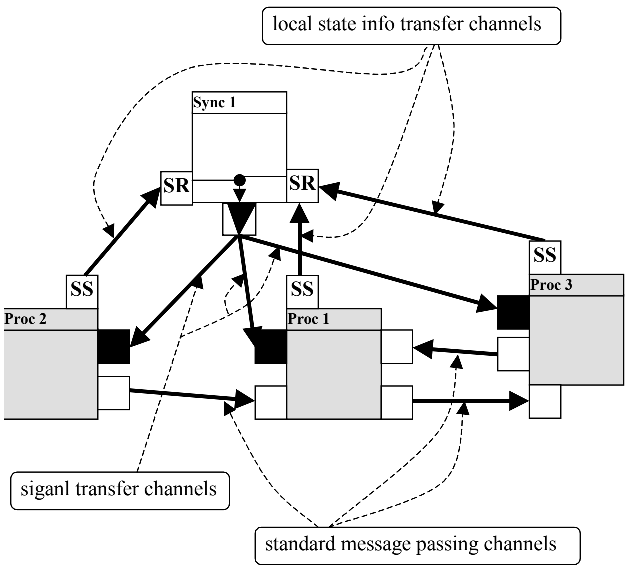
local state info transfer channels
Sync 1
SR
SR
SS
SS
SS
Proc 2
Proc 1
Proc 3
siganl transfer channels
standard message passing channels

Fig. 6 – Application level window – three processes with a synchronizer

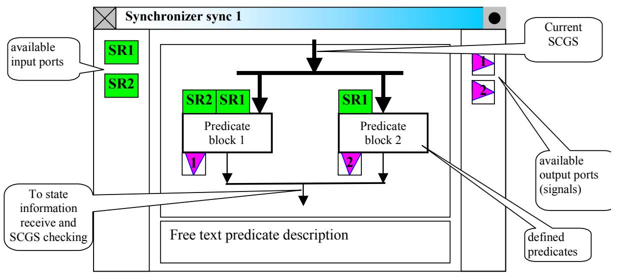

Fig. 7 – A synchronizer window

For a large number of processes, and for complex predicates, the amount of computations and communication a synchronizer have to perform can be problematic. There is a simple way to decentralize the synchronization control and to improve efficiency. It is by introduction of many synchronizers, each one responsible for a separate synchronization task. Moreover, synchronizers can be organized into hierarchies. Application processes can be split into groups. Each group can cooperate with its own synchronizer that can be connected to a higher-level synchronizer, Fig.9. There can be many levels in the hierarchy. Higher-level synchronizers act in the same way as low-level synchronizers. Lower level synchronizers send state messages to their higher lever synchronizers. A state message is send up in the hierarchy as a result of a predicate evaluation by a lower level synchronizer. A higher-level synchronizer computes SCGSs of subordinate synchronizers. It can know which predicates at the subordinate synchronizers are satisfied. Higher-level predicates can be constructed based on that knowledge and their evaluation can produce synchronization signals. The signals can be propagated to lower levels until they reach application processes. In such a way predicate evaluation and synchronization signal communication are performed in a parallel and distributed way.

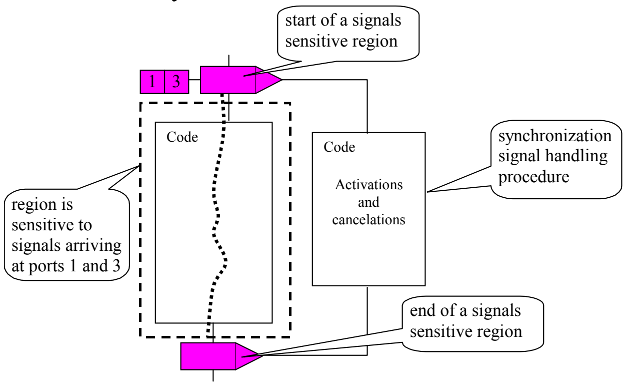
start of a signals sensitive region
1
3
Code
Code
Activations and cancelations
synchronization signal handling procedure
region is sensitive to signals arriving at ports 1 and 3
end of a signals sensitive region

Fig. 8 – Flow diagram sensitive to synchronization signals.

GRADE produces C programs based on standard PVM/MPI libraries. In general, such programs cannot be asynchronously interrupted to run a procedure and then to resume previous actions [BKT03]. MPI, PVM, and numerous standard C library procedures are not re-entrant. An application process cannot always react instantaneously on incoming signals. We propose to mark sections of the GRADE process control flow diagram as freely interruptible, uninterruptible or interruptible at defined points only. If an interruption cannot take place right away, a flag is set, and the signal is to be handled at the first opportunity in the future.

4. PRACTICAL IMPLEMENTATION SOLUTIONS

Practical implementation of the proposed system implies the following additions to existing GRADE.

a) synchronization of clocks in processors which execute application programs
b) state message exchange with timestamps
c) detection of consistent global states in processors
d) programming asynchronous process reaction to synchronization signals
e) extension of the graphical interface in existing GRADE.

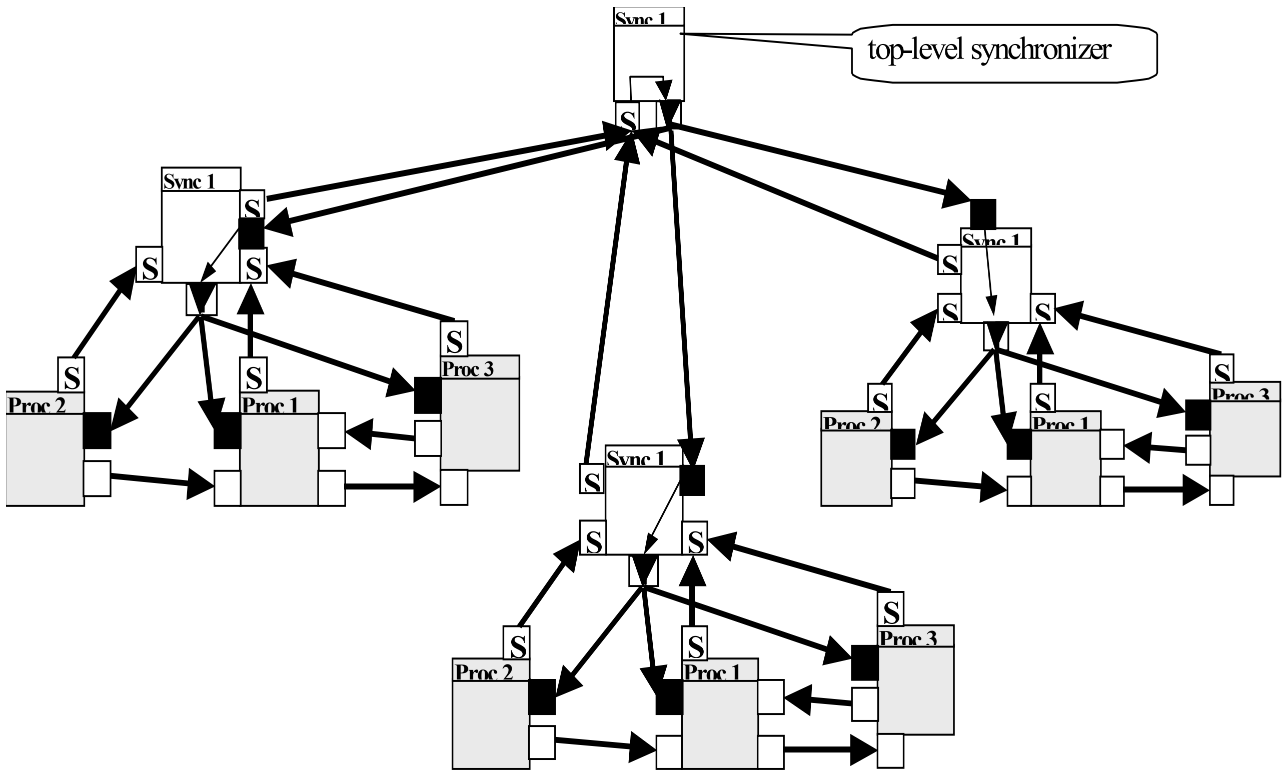

Fig. 9 – A hierarchic structure of synchronizers

For clock synchronization the Network Time Protocol [RFC] will be used, as immediately available. Fast Ethernet is used in our platform and we expect to synchronize the clocks with tolerance about 50 μs, so to be able to detect SCGS lasting at least twice that long. Given current CPU speeds, much can happen within that time period. Another solution is planned to be used in the future. It is the RBS protocol [EGE02] or hardware counters based on PCI counter cards controlled in a global way. Then clock tolerance of few microseconds can be achieved.

The timestamps can be easily introduced within GRADE communication library. A SCGS detection algorithm planned to be used is described in [S97, S00]. The algorithm waits for current process state termination before taking it into account. We want to eliminate waiting for state termination. A watch-dog mechanism in a synchronizer can solve the problem. When a process state starts (after a change), the synchronizer sets a timer. If no message with the state termination arrives soon, the timer triggers a dummy state termination. When the real termination comes, a dummy one is simply replaced by the real. Timer delay is set to 2ε+Dmax-Dmin, where ε is the accuracy of clock synchronization, Dmax and Dmin are the maximal and minimal message transfer times, respectively. This way dummy termination will be issued only for states lasting at least 2ε, i.e. considered stable enough to be considered. Dmin and Dmax are determined experimentally. Synchronizer monitors current transfer times and it can adjust the Dmax value.

Signals arriving at a process should trigger the handling procedure immediately. Standard message passing does not offer relevant mechanisms. The functionality of Active Messages [M98] is useful, but they provide a low-level implementation and need advanced network hardware (e.g. Myrinet). PVM message handlers [G97] are at a higher level, however, they are not triggered immediately upon a message arrival and they are not helpful for cancellation implementation, either. UNIX Real-Time signals can provide necessary features [B01, BKT03]. Within one system they can be delivered and handled within a few microseconds, while message passing and dispatcher processes can be used to transfer them between computers. The time between a process reports its new state to a synchronizer and receives a synchronization signal can be then estimated as twice the message transfer time + watch-dog timer delay + SCGS detection algorithm runtime. This value determines parallel task synchronisation granularity, which can be managed effectively by a synchronizer in our environment. In a FastEthernet cluster based on LAM MPI VIA [BPR01] the transfer time for a message up to 32 byte long is below 70 μs. In this environment, we expect the average synchronisation response time to be around 450 μs. The granularity becomes much finer with an introduction of a faster network. The use of GigaBitEthernet network based on zero-copy MPI EMP [ACP01] provides latency of 23 µs for transfers of messages of 1.5 KB. In this case the obtained reaction time will be shorter than 180 µs. The use of Myrinet 2000 network with MPI provides short-message latency (up to 100 bytes long) of 8.5 µs [My03]. In this case, we expect the synchronisation response time with MPI communication to be below 100 µs i.e. much better than that of GigaBitEthernet. The use of very fast DIMNET network [Tal02] can provide the net short message transfer time of less than 1.2 µs. One can expect to be able to decrease the MPI synchronisation response time to around 10 µs. An ideal solution here would be to use a separate network dedicated for synchronization and control purposes, as in CRAY T3E system [SC96]. In this case, one can expect the synchronisation response time decreased below 2 µs. A more up-to-date dedicated controller for synchronisation/ communication in cluster systems has been proposed in [HS00]. It enables hardware implementation of fuzzy synchronization [RG89] inside code regions in parallel application programs with a latency of 1.2 µs for 16 processes. An example of implementation of the distributed synchronization as a dedicated hardware has been described in [SW95]. With 100 MHz processors, it provides 200 ns latency for 256 processes in a barrier. It shows the potential of hardware distributed implementation of synchronization primitives.

5. CONCLUSIONS

A parallel programming environment which combines the standard message-passing paradigm with an advance synchronization and control model based on application global state predicates has been presented in this paper. Global predicates are used to control application program execution together with asynchronous activation and cancellation mechanism. Data transmissions are de-coupled from synchronization and relevant control infrastructure in programs. The synchronization and control code is well separated from the proper application computational code and is easy to understand and verify. Global predicates can implement application control and synchronization, which is correct by construction. The paper contains a discussion of the implementation problems and gives viable solutions.

The proposed programs execution control method will be implemented as a user-friendly parallel program graphical design system. Such a system is currently under work by enhancing an existing GRADE graphical parallel programming environment. This project is implemented within a co-operation with the SZTAKI Institute in Budapest who has been a designer of the GRADE system. Some of proposed elements do not exist in current software environments, e.g. asynchronous activation/cancellation, and will constitute an original extension of current programming methodology. Structural design of the control interface in our proposal is close to GRADE user interface philosophy, which will facilitate their integration. We hope the integrated system will be easily understandable to a programmer and will enable to him a new more useful style of parallel program design. An operational system that we will have completed soon will let us practical evaluation of all proposed solutions.

This work has been sponsored by the KBN Grant N.4T11C007 22. and by internal grants of the PJIIT.

6. REFERENCES

[ACP01] M. Apte, S. Chakravarthi, J. Padmanabhan and A. Skjellum, A Synchronized Real-Time Linux Based Myrinet Cluster for Deterministic High Performance Computing and MPI/RT, Ninth International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2001), April 2001, San Francisco.

[B00] J. Borkowski, Towards More Powerful and Flexible Synchronization Primitives, in Proc. of Inter. Conf. on Parallel Computing in Electrical Engineering PARELEC 2000, August 2000, Trois-Rivieres, Canada. IEEE PR00759, pp.18-22.

[B01] J. Borkowski, Interrupt and Cancellation as Synchronization Methods, in Proc of 4th Int. Conf. Parallel Processing and Applied Mathematics PPAM 2001, Nałęczów, Poland, LNCS 2328, Springer 2001.

[BKT02] J. Borkowski, D. Kopański, M. Tudruj, „Adding Advanced Synchronization to Processes in GRADE", in Proceedings of the Int. Conf. on Parallel Processing and Electrical Engineering PARELEC 2002, Warsaw, Poland, IEEE 2002.

[BKT03] J. Borkowski, D. Kopański, M. Tudruj, Implementing Control in Parallel Programs by Synchronization-Driven Activation and Cancelation, Proc. of the 11-th Euromicro PDP '03, Feb. 2003, Genova, Italy, IEEE 2003.

[BPR01] M. Bertozzi, M. Panella, M. Reggiani, Design of a VIA Based Communication Protocol for LAM/MPI Suite, Ninth Euromicro Workshop on Parallel and Distributed Processing (PDP '01) ,February 07 - 09, 2001, pp. 27-33.

[CM91] R. Cooper and K. Marzullo, "Consistent detection of global predicates, "Proceedings ACM/ONR Workshop on Parallel Distributed Debugging, pages 163-173, 1991.

[CO94] Cohen W.E., Dietz H.G., Sponaugle, Dynamic Barrier Architecture for Multi-Mode Fine-Grain Parallelism Using Conventional Processors, 1994 Int. Conf. on Parallel Processing, pp. I 93-96.

[DK99] D. Drótós, P. Kacsuk, GRAPNEL To C Translation in the GRADE Environment, Computers and Artificial Intelligence, Vol. 18, No. 4. pp. 415-424, 1999.

[EGE02] J. Elson, L. Girod and D. Estrin, Fine-grained network time synchronization using reference broadcasts, Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachussetts, USA, December 2002.

[FR95] Eddy Fromentin and Michel Raynal, Characterizing and detecting the set of global states seen by all observers of a distributed computation, Proceedings of the Fifteenth International Conference on Distributed Computing Systems, pp. 431-438, 1995.

[G97] Al Geist Advanced Tutorial on PVM 3.4 New Features and Capabilities, http://www.csm.ornl.gov/pvm/EuroPVM97/

[GM01] V. K. Garg and N. Mittal. On Slicing a Distributed Computation, Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS), pages 322-329, Phoenix, Arizona, April 2001.

[GW94] Detection of weak unstable predicates in Distributed programs, V.K Garg, B. Waldecker, IEEE Transactions on Parallel and Distributed Systems, 5(3), pp. 299--307, March 1994.

[GW96] V. K. Garg, B. Waldecker, Detection of Strong Unstable Predicates in Distributed Programs, IEEE Trans. on Parallel and Distrib. Systems, Vol. 7, No. 12, December 1996, pp. 1323-1333.

[HS00] K. Hyakawa, S. Sekiguchi, Design and Implementation of a Synchronization and Communication Controller for Cluster Computing Systems, 4-th Int. Conference on High Performance Computing in Asia-Pacific Region, Vol. 1, May 2000, pp. 76- 81.

[KDF97] Kacsuk, P., Dózsa, G. and Fadgyas, T., GRADE: A Graphical Programming Environment for PVM Applications Proc. of the 5th Euromicro Workshop on Parallel and Distributed Processing, London, 1997, pp. 358-365.

[KDFL99] The GRED Graphical Editor for the GRADE Parallel Program Development Environment P .Kacsuk, G. Dózsa, T. Fadgyas and R. Lovas Future Generation Computer Systems, No. 15 (1999), pp. 443-452.

[M89] F. Mattern. "Virtual Time and Global States in Distributed Systems". Proc. Workshop on Parallel and Distributed, Algorithms, Chateau de Bonas, Oct. 1988, M. Cosnard et al. (eds.), Elsevier / North Holland, pp. 215-226, 1989.

[M95] Mark Minas, Detecting Quantified Global Predicates in Parallel Programs, Europar 95 , Stockholm, Sweden. Proceedings. Lecture Notes in Computer Science, Vol. 966, Springer, pp. 403-414.

[M98] P. J. Mucci, "An Efficient Transport Independent Active Messaging Implementation for PVM", Technical Report UT-CS-98-399, 1998, http://citeseer.nj.nec.com/93955.html

[My03] Myricom Corp. GM 1.6.4 API Performance with PCI64B and PCI64C Myrinet/PCI Interfaces, April 2003, http://www.myri.com/myrinet/performance/index.html

[OL95] Olnovitch, H.T., ALLNODE Barrier Synchronization Network, 9-th Int. Parallel Processing Symposium, April, 1995, pp. 265-269.

[PGR] The P-GRADE Visual Parallel Programming Environment, http://www.lpds.sztaki.hu/teaching_materials/P-GRADE/index.htm

[RFC] Request for Comment RFC1305 Network Time Protocol (Version 3) Specification, Implementation and Analysis.

[RG89] R. Gupta, The Fuzzy Barrier: A Mechanizm for High Speed Synchronization of Processors, Proc. of the 3rd ASPLOS Conference, April 1989, pp. 54-63.

[S00] Scott D. Stoller: "Detecting Global Predicates in Distributed Systems with Clocks". Distributed Computing, Volume 13 Issue 2 (2000) pp 85-98.

[S97] S.D. Stoller, "Detecting Global Predicates in Distributed Systems with Clocks". Proc. 11th International Workshop on Distributed Algorithms (WDAG 97). Lecture Notes in Computer Science, Springer-Verlag, 1997.

[SC96] Scott S. L., Synchronization and Communication in the T3E Multiprocessor, Proceedings of the 7-th ASPLOS Conference, 1996, pp. 26-36.

[SW95] S. Shang, K. Hwang, Distributed Hardwired Barrier Synchronization for Scalable Multiprocessor Clusters, IEEE Trans. On Parallel and Distributed Systems, vol. 6, June 1995, pp. 591 – 605.

[SWP01] P. Shivam, P. Wyckoff, D. Panda, EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing, Proceedings of Conference on High Performance Networking and Computing, Denver, Colorado, Nov. '01,pp. 57 - 57

[Tal02] N. Tanabe et al., Low Latency Communication on DIMMnet-1 Network Interface Plugged into a DIMM Slot, Proceedings of the Int. Conf. on Parallel Computing in Electrical Eng., Warsaw, Sept. 2002, pp. 9 – 14.

[TG98] A. Tarafdar and V.K. Garg. Predicate Control for Active Debugging of Distributed

Programs. Symposium on Distributed and Parallel Debugging, 1998.

[KT98] M. Tudruj, P. Kacsuk, Extending Grade Towards Explicit Process Synchronization in Parallel Programs, Computers and Artificial Intelligence, vol 17, 1998, No. 5 pp 507-516.

__Janusz Borkowski__ is a PhD candidate and an assistant in the Chair of Parallel Computing in the Polish-Japanese Institute of Information Technology in Warsaw. He has got a MSc degree from the Department of Mathematics, Mechanics and Informatics of Warsaw University in 1995. He teaches parallel programming methods, network programming, system programming. His research interests cover control and synchronization methods in parallel programs, parallel simulation, computation result visualization in parallel system. He is the author or co-author of 15 research papers published in proceedings of scientific conferences.

__Marek Tudruj__ is a professor and a head of the Chair of Parallel Computing in the Polish-Japanese Institute of Information Technology in Warsaw. He is also an associate professor and a head of the Computer Architecture Group in the Institute of Computer Science of the Polish Academy of Sciences in Warsaw. He graduated in 1967 from Electronics Department of the Warsaw University of Technology. He has got a PhD and DSc degrees from the Institute of Computer Science of the Polish Academy of Sciences in Warsaw in 1979 and 1992, respectively. He teaches computer architecture, parallel computing systems methodology, parallel systems modeling methods. His current research interests cover parallel systems architecture, execution control in parallel programs, parallel program optimization, support tools for parallel programming. He is the author or co-author of more than 70 research papers published in journals and proceedings of scientific conferences.

__Damian Kopanski__ is the systems administrator and a MSc student in the Chair of Parallel Computing in the Polish-Japanese Institute of Information Technology in Warsaw. He studied at the Faculty of Electronics and Information Technology of Warsaw University of Technology. In 2001 he graduated with the Engineer Degree from the Polish-Japanese Institute of Information Technology. His thesis is concerned with the implementation of the graphical parallel program design system GRADE with synchronization primitives based on application program global states. He is a co-author of 4 research papers published in proceedings of scientific conferences.