# An Execution Order Control Method of Distributed Processes for Sharing Global Variables

Koichi ASAKURA, Toyohide WATANABE and Noboru SUGIE

Department of Information Engineering,
Faculty of Engineering, Nagoya University
Furo-cho, Chikusa-ku Nagoya 464-01, JAPAN

## Abstract

*The parallelizing compiler divides a given program into several processes which can be executed simultaneously in distributed computer environment. However, a program has global variables generally. Therefore, the processes have to share global variables since they are common in the program. In order to manage global variables correctly, the processes need to possess the broadcasting facility for data sharing and the synchronization facility for execution order control among the processes. In this paper, we proposed execution order control methods among the processes for sharing global variables on the distributed computer environment.*

## 1 Introduction

Today, the distributed computer environment, in which several autonomous workstations and computing facilities are connected organically through the local area network, is becoming common. The software support for parallel processing is required in order to control such computer systems efficiently. A parallelizing compiler is one of the software support products, which can solve such requirements. The parallelizing compiler divides a given program automatically into several program fragments which can be executed simultaneously on different computers[1,2]. Therefore, end-users are free from complicated problems such as load balancing methods and synchronizing means. However, the computing facilities for efficient parallel processing such as a shared memory and a high speed communication channel, which are always equipped in many of the traditional parallel computing systems[3,4], are not always supplied in the distributed computer environments. The lack of such hardware support makes it impossible to apply the parallelizing techniques such as DO loop parallelizing methods[5–8], which have been developed in the super-computers and vector-computers successfully, to distributed computer environments directly as they were.

In order to solve the above problem, we proposed the function call parallelizing method[9], which makes each function call statement execute simultaneously. Each function in C programs has its own variable space individually, and the data transfer between these functions is performed only as parameters when the function is invoked and as a return value when the execution of the function is finished. Therefore, the function call parallelizing method is more suitable to the distributed computer environment than the DO loop parallelizing method from a viewpoint of the interprocess communication issue.

However, we have not solved some programming language-dependent problems. One of them is the data sharing control mechanism among distributed processes. A program uses global variables generally. Global variables are commonly usable objects among processes (or program fragments) to implement the data sharing mechanism. As described above, the distributed computer environment has no peculiar hardware equipment for data sharing. Thus, we have to achieve the data sharing facility with respect to interprocess communication.

In this paper, we address practical data sharing methods based on the interprocess communication in distributed computer environments. Our basic data sharing strategy is to calculate the execution order among functions firstly and then to control concurrently executable processes by the synchronization indicator. This synchronization indicator is called "trigger." The unique triggers are assigned to individual processes. A process informs other processes of its own execution end through broadcasting of this trigger. Therefore, the interdependent process can execute soon after having accepted the trigger. The trigger makes the execution order control of processes possible without decrease of effectivity for parallel processing.

The rest of this paper is organized as follows. Section 2 summaries the related work and proposes our methods comparatively. Section 3 expresses the processing procedures of our methods and gives the brief examples. In Section 4, we evaluate our methods through some experiments. Finally, the conclusion and future work are arranged in Section 5.

## 2 Related Work and Our Strategy

The mature consideration of network-wide data sharing is required for efficient parallel processing in the distributed computer environment. It is important for end-users to be able to develop application programs without paying any peculiar attentions to the configuration of computer systems. Many researches are taken place for support of parallel processing, especially data sharing, until today. Linda introduces new concepts for data sharing: Tuple and Tuple Space[10]. In programming with Linda, all operations for parallel processing such as creation of new processes, communication between processes, data sharing among processes and so on, are accomplished through Tuple Space. In Tuple Space, processes and data are treated as Tuples. Additionally, all processes share Tuple Space for data sharing and interprocess communication. Namely, the operations to Tuple Space enables end-users to achieve data sharing unconsciously. On the other hand, C*, which was developed as C-like parallel programming language for the hypercube multiprocessor system, proposes the notion of "virtual processor" as a task unit[11]. Virtual processors are mapped dynamically to the physical processors. Addi-

tionally, the data sharing among processes is managed by "global name space." The virtual processor can refer to variables in another virtual processor through the global name space, and this allows end-users to pay no attentions to controlling communication between processes.

Although these parallelizing approaches are successful in data sharing mechanism, they are too heavily related to processing methods in the central processing system. Namely, their design methodologies are dependent on tightly-coupled computer architectures. For example, the computing system needs a shared memory for the efficient implementation of Tuple Space. Although the implementation of Linda in the distributed memory multiprocessor system has been reported[12], the approach is architecture-oriented and is not appropriate for distributed computer environments. Also, C* is never effective because the computer configuration in C* needs a high speed communication facility for the efficient implementation of global name spaces.

The following requirements are notable points for making parallel processing successful in the distributed computer environment:

- to avoid sharing data to be commonly accessed among activated processes;
- to reduce the amount of communication among processes.

These requirements are dependent on the fact that the distributed computer environment has a disadvantage of communication capacity among processors because independent workstations are connected loosely by the local area network. As for these requirements, we adopted the function call parallelizing method in our parallelizing compiler. The function call parallelization makes the function call statements execute in parallel. Each function has its own variable space, which reduces the data to be shared among processes. Additionally, the granularity of processes generated by the function call parallelizing method is coarser than that created by the traditional DO loop parallelizing method, and as a result our parallelizing method decreases the frequency and account of communication among processes. Therefore, our function call parallelizing strategy is suitable for distributed computer environments.

However, a program often includes global variables which must be shared among distributed processes. As several processes in different processors may refer to or update global variables that are unique and common in the program, it is important to reduce the access frequency and account of communication data. Our data sharing method is enhanced on the basis of the function call parallelization paradigm[9] with respect to the access control of global variables. Our basic idea is the introduction of the synchronous control indicator. This indicator is called as the trigger, which is assigned to each function. When functions refer to or update the same data, they have constraints for their execution order to ensure the correctness of the computation result. The execution order of critical program regions is safely controlled by the synchronization mechanism with trigger. A process broadcasts its own trigger to other processes at the execution end of the critical program region in order to allow the interdependent processes to execute the critical program region successively. Our data sharing method with the trigger has the following features.

1. The exclusive execution in the critical program regions is ensured. Therefore, when a process updates a global variable, the process informs other processes of the new value for the variable at once before other processes execute the critical program region. Namely, the process does not have to inform even when the process updates the variable. Thus, our method reduces the frequency of communication.

2. It is not always possible to detect the full execution order among individual functions. In our method, the full execution order of all functions is not needed. Namely, our synchronization method can be applied to the program even in case that the partial execution order was only calculated.

# 3 Data Sharing Method

## 3.1 Preliminaries

We explain first of all a practical way to parallelize programs before attaching to our data sharing method. Our parallelization strategy is based on the function-specific program division paradigm. If it was detected that two different functions (such as calling function and called function) are executable simultaneously, the following procedure is applied to them in order to accomplish to the high parallel execution:

1. Extract relocatable program regions for the function call statement from statement sequence of the program;
2. Move the function call statement to the upper location of the relocatable program region;
3. Insert the synchronization statement into the lower location of the region, if necessary.

Here, the relocatable program region is defined as follows.

### Relocatable Program Region

There are statements $S_i$'s $(i = 1, 2, \cdots)$. $S_i$ is executed before $S_j$ if $i < j$. The relocatable program region of $S_i$ is $[S_m, S_n]$, where

1. $m < i < n$.
2. $S_i$ has interdependency relationship with $S_m$ and $S_n$.
3. $S_i$ is independent of $S_l$ such that $^{\forall}l, m < l < i, i < l < n$. □

The interdependency relationship is collected in the variable analysis phase[13,14]. For example, let us find the relocatable program region of $S_2$ in Figure 1. Since $S_2$ updates a variable a, it has an interdependency relationship with the statements that refer to or update the same variable: $S_1$ and $S_6$. Thus, the relocatable program region of $S_2$ is $[S_1, S_6]$.

The relocatable program region shows the degree of parallel execution for the function in $S_i$. The larger the relocatable program region of the call statement is, the more effective the degree of parallel execution becomes[9].

```
S1: b = a + 1;
S2: a = func1();
S3: b = b + 1;
S4: c = func2(b);
S5: d = b + e;
S6: f = a + c;
```

Figure 1: An example of program fragment

## 3.2 Advanced Variable Analysis Method

The above example program includes two function call statements: $S_2$ and $S_4$. The relocatable program region of $S_2$ is $[S_1, S_6]$. Also, $[S_3, S_6]$ is in $S_4$[1]. Therefore, two functions `func1` and `func2` are executable in parallel. Namely, `func1` and `func2` are organized as different processes on the basis of the function-based parallelizing method. However, let us consider that these two functions manipulate the same global variables. Figure 2 shows this case. In Figure 2, the variable g is a global variable which is common among all functions. Since our previous variable analysis procedure does not pay attention to the variables to be manipulated within functions, the interdependency relationship between the statements to manipulate global variables, which are included in functions, is disregarded. Therefore, if the same global variables are referred simultaneously from different processes, the execution result is not assured.

In order to deal with this problem, we must improve our previous variable analysis procedure. Namely, the analysis procedure of function call statements must be applied to not only the parameters of functions, but also the variables that are manipulated in functions[15]. This enhancement makes it possible that the actual interdependency relationship among every variable is calculated. When this advanced variable analysis procedure is applied to the above example, the relocatable program region of $S_2$ is $[S_1, S_4]$ since the interdependency relationship between $S_2$ and $S_4$ is assigned in Figure 1. Therefore, `func1` and `func2` are executed sequentially, so that they make no troubles for the global variable operations.

This advanced variable analysis procedure is based on a very simple idea conceptually. We call this method the analysis-based method, hereafter. However, this method may decrease the possibility of efficiency of parallel execution. The execution of `func2` is never started before the whole execution of `func1` has finished, although almost all the statements (that are omitted statements with dots in Figure 2) in `func1` and `func2` can be executed simultaneously.

## 3.3 Data Sharing Method with Trigger

The analysis-based method has the problem that the effective parallel execution may be disturbed. This problem is derived from the fact that the analysis-based method does not provide the exact synchronization control among distributed processes. Therefore, the maximum efficiency of parallel execution can be accomplished if and only if the exact synchronization control method is achieved. From this consideration, we developed a new data sharing method with the synchronization indicator: "trigger."

[1]There is no dependency relationship between $S_4$ and $S_5$ although two statements use the same variable b. This is because the parameters are passed by the value in C language and $S_4$ refers to only the value of b.

```
int func1()                    int func2()
{                              {
      .....                          .....
      .....                          .....
      .....                  S2:   g = a * 2;
      .....                        }
S1:   g = a + b;
}
```

g is a global variable.

Figure 2: Interdependent functions

Broadcasting of the trigger to other processes means that the execution of the critical program region that manipulates global variables has finished. The trigger allows other interdependent processes to execute the critical program region in sequence. The actual strategy is as follows:

1. Calculate the execution order of function in the program analysis phase. Additionally, the execution order of statements which manipulate global variables is determined;
2. Assign a trigger to each statement which manipulates global variables;
3. Insert the statement, which broadcasts the trigger, into the lower location of the statement that manipulates the global variable;
4. Insert the synchronization statement, which checks up whether the request trigger has already been broadcasted, into the upper location.

The trigger is sent to individual processes in all network-connected workstations by the broadcasting facility.

Figure 3 shows our trigger mechanism for synchronization between `func1` and `func2` in Figure 2. As shown in Figure 1, the execution order of these functions is "`func1` → `func2`." Thus, the execution order of statements is "$S_1 \rightarrow S_2$" in Figure 3. Therefore, in `func1` the synchronization statement "`trigger("func1-S1-g");`" that broadcasts the trigger "`func1-S1-g`" (,which means that this trigger is assigned to the statement $S_1$ in `func1` for the variable g) is inserted into the lower location of the statement which operates the global variable. On the other hand, in `func2` the synchronization statement "`wait_trigger("func1-S1-g");`" is inserted into the upper location of the statement that updates the variable g in order to detect the broadcasted trigger "`func1-S1-g`". These two synchronization statements make both the execution order of the critical program region among two processes and the parallel execution of two functions successful.

```
int func1()                    int func2()
{                              {
      .....                          .....
      .....                          .....
      .....                    wait_trigger("func1-S1-g");
      .....                  S2: g = a * 2;
S1:   g = a + b;               trigger("func2-S2-g");
      trigger("func1-S1-g");   }
}
```

Figure 3: Synchronizing procedure with trigger

In this method, the check on whether the statement which manipulates global variables can be executed or not

is accomplished immediately before the statement starts to execute. Therefore, this method does never decrease the efficiency of parallel execution as the analysis-based method does. Figure 4 illustrates the execution image of three functions under the execution order control in comparison with these two methods. The trigger-based method provides more efficient parallel execution than the analysis-based method. However, the trigger-based method may take much time to control the trigger and this control overhead might cancel the effects of parallelized functions. Additionally, this trigger-based method is not applied in case that the execution order of statements which operate global variables can not be detected explicitly. Although the trigger-based method is basically used in our compiler, the analysis-based method is applied when the trigger-based method can not be adaptable because of the above drawback.

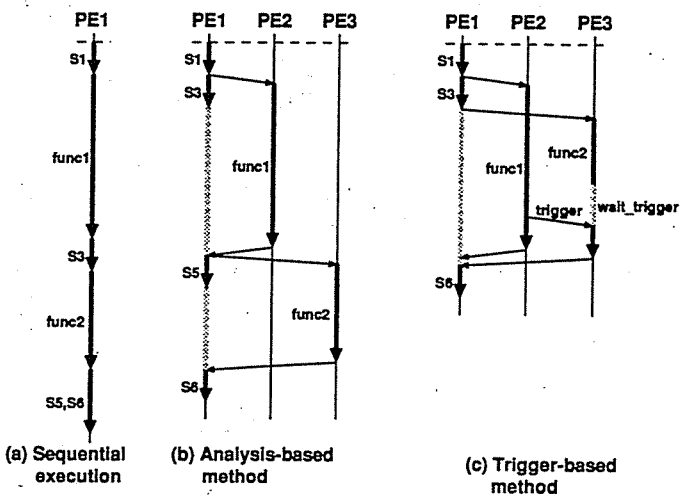


Figure 4: Execution image

## 4 Experiment

In order to evaluate our data sharing methods, we make some experiments. We consider the subset-sum problem as an example program. The subset-sum problem is NP-complete and takes much time to get solutions. Therefore, it is effective if this program can be executed in parallel. This program receives back the name of the file in which the element numbers are written and the goal value. In this program, global variables are utilized in order to store answers. Each distributed process calculates one candidate answer from a given data. Global variables are updated when the candidate answer is closer to the goal value than the totally computed answer. In our distributed computer environment, there are 8 SUN-SPARCstation/SLC workstations connected by the local area network (Ethernet).

Figure 5 compares the times to execute three versions of the program: the original sequential program, the program parallelized by the analysis-based method and the program parallelized by the trigger-based method. The horizontal axis shows the number of elements in the given set. The program parallelized by the analysis-based method takes more time than the program parallelized by the trigger-based method. This is because some distributed processes created by the analysis-based method can not be executed in parallel for execution control. On the other hand, the

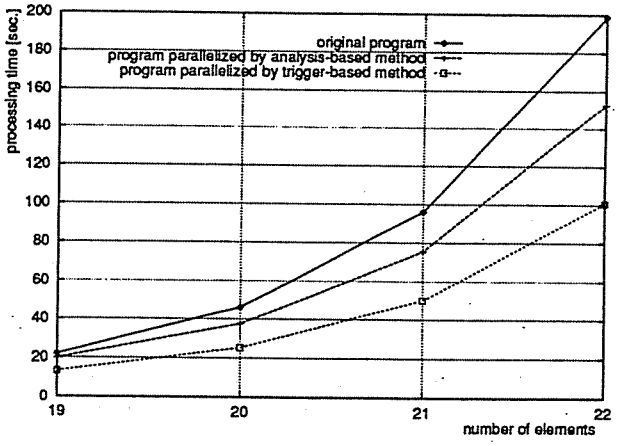


Figure 5: Experimental result 1

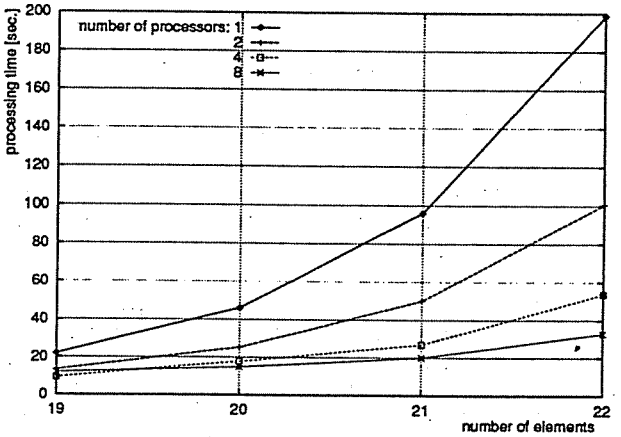


Figure 6: Experimental result 2

trigger-based method has no constraints for execution control and all processes could be executed in parallel. Thus, the effectivity of our trigger-based method can be proved clearly through this experiment.

Figure 6 shows the processing times of the program on several experimental environments: 1, 2, 4 and 8 processors, respectively. This result expresses that our method is sensitive and adaptable to the various configuration of the system. However, the execution time of 8 processors is larger than the execution time of 4 processors in case that the number of elements is 19. This is because that the communication cost in the distributed computer environment is large in 8 processors by comparing with 4 processors.

## 5 Conclusion

In this paper, we reported the method for dealing with global variables among processes in distributed computer environments. Two actual methods are proposed: one is achieved by the advanced variable analysis method, and another is accomplished by the trigger control mechanism. These methods have the following features.

- The data sharing procedure is based on the broadcasting facility and execution order control mechanism. The control mechanism of execution order among processes ensures the exclusive execution in the critical program region. Therefore, each process informs all

other processes of the update-permission event just at once before the synchronization statement is executed. This method can decrease the frequency of communication among distributed processes.

- The trigger-based method makes more precious exclusive execution control of the critical program region possible than the analysis-based method. Namely, the trigger-based method can accomplish the maximum efficiency of parallel execution.

- The execution order analysis of functions and statements which manipulate global variables is necessary in our method. However, our method can be applied to even program fragments whose full execution order is not determined. Namely, our method can deal with program fragments whose partial execution order is only calculated.

Additionally, the experimental result made it clear that our method is effective in the distributed computer environment.

Finally, we report our future work as follows.

1. The trigger-based method takes much time, and it may violates the efficient parallel execution. Therefore, we must pay attention to the trade-off problem between the extra processing time for treating a trigger and shortened execution time for parallel processing.

2. Figure 6 explains that the communication procedure among distributed processes takes much time. Since each process has global variables, the event of updating global variables must be sent to all processes. This control mechanism becomes high cost processing. In order to deal with this drawback, we are considering the special process for managing global variables. This process manages all global variables and the distributed processes communicate with this process for referring to and updating the global variables.

3. The information about execution order among distributed processes is necessary for our method. Namely, our method does not take the dynamic behavior of a program into consideration. Therefore, our method can be applied in case that the calling relationship between functions does not change dynamically during the execution of the program. We must consider a method which can manage the dynamic behavior of programs.

4. The C language has a peculiar type of variables: pointer variables. Pointer variables make the sharing data among processes possible as well as global variables. Also, we can refer to the value of a variable through the pointer variables (aliasing). We have to deal with pointer variables in the same manner as global variables.

## Acknowledgements

We are very grateful to Prof. T. FUKUMURA of Chukyo University, and Prof. Y. INAGAKI and Prof. J. TORIWAKI of Nagoya University for their perspective remarks, and also wish to thank Ms. K. SUGINO and our research members for their many discussions and cooperations.

## References

[1] C. D. Polychronopoulos, M. Girkar, M. R. Haghighat, C. L. Lee, B. Leung and D. Schouten: "Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing and Scheduling Programs on Multiprocessors," *Proc. of the 1989 Int'l Conf. on Parallel Processing*, Vol.2, pp.39–48 (1989).

[2] H. Honda, S. Mizuno, H. Kasahara and S. Narita: "Parallel Processing Scheme of a Basic Block in a Fortran Program on OSCAR," *Trans. on IEICE of Japan*, Vol. J73-D-I, No.9, pp.756–766 (1990) [in Japanese].

[3] D. J. Kuck, E. S. Davidson, D. H. Lawrie and A. H. Sameh: "Parallel Supercomputing Today and the Cedar Approach," *Trans. on IEICE of Japan*, Vol.J71-D, No.8, pp.1361–1374 (1988).

[4] H. Kasahara, S. Narita and S. Hashimoto: "Architecture of OSCAR – Optimally Scheduled Advanced Multiprocessor – ," *Trans. on IEICE of Japan*, Vol. J71-D, No.8, pp.1440–1445 (1988) [in Japanese].

[5] D. A. Padua and M. J. Wolfe: "Advanced Compiler Optimizations for Supercomputers," *Comm. of the ACM*, Vol.29, No.12, pp.1184–1201 (1986).

[6] R. Cytron: "Doacross: Beyond Vectorization for Multiprocessors (Extended Abstract)," *Proc. of the 1986 Int'l Conf. on Parallel Processing*, pp.836–844 (1986).

[7] T. Marushima and Y. Muraoka: "Parallel Processing Method for Iterative Loops : doalong," *Trans. on IEICE of Japan*, Vol.J71-D, No.8, pp.1511–1517 (1988) [in Japanese].

[8] S. P. Midkiff and D. A. Padua: "Compiler Generated Synchronization for Do Loops," *Proc. of the 1986 Int'l Conf. on Parallel Processing*, pp.544–551 (1986).

[9] K. Asakura, T. Watanabe and N. Sugie: "C parallelizing Compiler on Local-network-based Computer Environment," *Porc. of the 7th Int'l Parallel Processing Symp.*, pp.849–853 (1993).

[10] N. Carriero and D. Gelernter: "Linda in Context," *Comm. of the ACM*, Vol.32, No.4 pp.444–458 (1989).

[11] P. J. Hatcher, A. J.Lapadula, R. R. Jones, M. J. Quinn and R. J. Anderson: "A Production-Quality C* Compiler for Hypercube Multicomputers," *Proc. of the 3rd ACM SIGPLAN Symposium on Principle & Practice of Parallel Programming*, pp.73–82 (1991).

[12] R. Cohen: "Optimising Linda Implementations for Distributed Memory Multicomputers," *Proc. of the 1st Annual Users' Meeting of Fujitsu Parallel Computing Research Facilities*, ANU-5 (1992).

[13] D. E. Maydan, J. L. Hennessy and M. S. Lam: "Efficient and Exact Data Dependence Analysis," *Proc. of the ACM SIGPLAN '91 Conf. on Programming Language Design and Implementation*, pp.1–14 (1991).

[14] G. Goff, K. Kennedy and C.-W. Tseng: "Practical Dependence Testing," *Proc. of the ACM SIGPLAN '91 Conf. on Programming Language Design and Implementation*, pp.15–29 (1991).

[15] P. Tang: "Exact Side Effects for Interprocedural Dependence Analysis," *Proc. of the 1st Annual Users' Meeting of Fujitsu Parallel Computing Research Facilities*, ANU-2 (1992).