Regular Paper

A Proposal for an Operating System Designed for Cluster Servers

KAZUYA TAGO*, YASUSHI NEGISHI* and MIKAKO HOSHIBA*

The cluster server is a collection of computers connected by a high speed link. It provides file services and job processing services to client workstations via LAN. The architecture can realize a wide range of scalability and low initial cost. The operating system of the server needs to provide a single system image by implementing a distribution transparent file system, distributed shared memory and inter-process communication. An approach to build an efficient basis for them is proposed in this paper. The approach improves the page data transfer performance and control data transfer performance by introducing an architectural support called Pion and a new kernel layer called CPI. Pion generates a data stream, which is handled by link hardware, from scattered data on memory. It allocates memory to the data stream automatically, this frees software from packet marshaling jobs.

CPI enables to fetch a remote page without invoking a server process. A file system using the support has been implemented based on OSF/1 operating system. It has been shown that over 80% of file page transfer cost is expected to be reduced compared with NFS by the approach used in CPI and Pion design.

1. Introduction

Microprocessors make various types of distributed processing and parallel processing systems feasible. The computing environment of today is built from a collection of a large number of processors and computers. Each computer is designed as a component of a hierarchy of a distributed processing system. It operates as a special purpose machine rather than a fully functioning general purpose one, and its design assumes cooperation with other computers. The strategy for integrating whole systems into a harmonized one and the technology to build each machine affect each other. System designers and researchers are requested to take them into account simultaneously.

The centralized processing by a large scale multiprocessor, distributed processing by high performance workstations and client-server computing are typical system configurations. The centralized processing provides high efficiency and the distributed processing provides a good man-machine interface. Client-server systems of low cost compact client workstations and powerful servers are expected to be able to realize both merits at lower cost.

An approach to design cluster servers for the client-server configuration is proposed in this paper. The cluster server is composed of existing computers combined by a high speed link. It is classified into the loosely coupled multiprocessor architecture. The configuration provides a wide range of performance scalability and low initial cost. On the other hand, the overhead caused by the inter-node communication becomes a problem and have a reduction is required.

We implemented a cluster server by connecting IBM PS/55[1] workstations. The operating system is based on OSF/1[2]. We designed and coded an inter-node communication mechanism with architectural support and a new file system. The mechanism transfers pages between file buffers of cluster nodes without building communication packets. It has been shown through measuring static steps of the file system that the overhead is expected to be reduced to a considerable extent in comparison with NFS[3]. The design strategy and performance estimation is stated in this paper.

2. Cluster Server

2.1 Client-Server Model

Figure 1 shows our target system configuration. A compact client workstation provides a man-machine interface including window management, command inter-

*Tokyo Research Laboratory, IBM, 102 Sanbancho 5-19, Chiyoda-ku, Tokyo Japan.

[1]PS/55 is a trademark of International Business Machines Corporation.
[2]OSF/1 is a trademark of Open Software Foundation, Inc.
[3]NFS are trademark of Sun Microsystems, Inc.

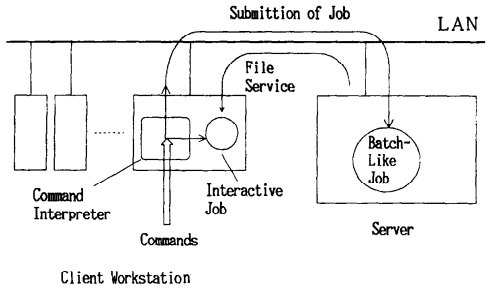

Fig. 1 A Client-Server System.

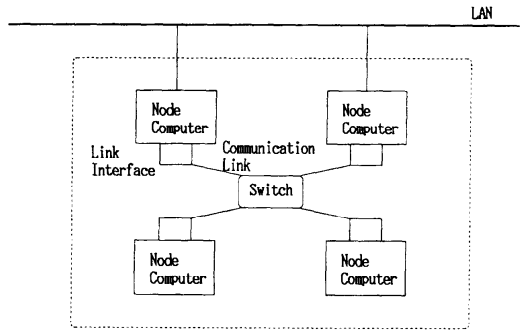

Fig. 2 A Cluster Server.

pretation, text editing, graphic interface and multimedia support. A server provides file management facilities, high throughput job processing facilities and parallel processing facilities such as numerical intensive computation (NIC).

A command interpreter on the client workstation decides the execution site of user commands and provides an integrated view of multiple server environment. When an user invokes a batch-like job such as program compilation, the command interpreter asks an appropriate server for its execution. When a user invokes an interactive job such as text editing, the command interpreter executes the command locally. The command interpreter is being implemented on the RT PC[1] workstation with UNIX[2] operating system.

Users of a distributed system of workstations of similar performance have attempted to execute parallel programs on the network and to perform load balancing by migrating processes [10]. However, LAN is not efficient for these purposes and does not provided sufficient protection mechanism from illegal access. For example, granting a resource access right to a remote kernel is associated with the process migration, and the grant between untrusted workstations via an untrusted medium is dangerous. Multiprocessor systems are able to perform them in a safer and more efficient way. End-users of client workstations are freed from troublesome jobs of managing data and programs on disk storage by centralized file management. On the other hand, it is hard to provide a good man-machine interface by centralized processing. The client-server system design is intended to provide high performance with a good interface by selecting job execution site based on its property.

2.2 Overview of The Cluster Server

The cluster server consists of a collection of existing computers as shown in Fig. 2. Node computers may be uniprocessor workstations or tightly coupled multiprocessor machines. Hardware resources of node computers, such as disks, are shared among all cluster nodes by the help of an operating system. The core architecture of nodes including processor, memory management system and cache management system is not modified from the original design. This makes the difference from a pure loosely coupled multiprocessor architecture. For example, Transputer[1] is designed as a node of multiprocessor from the start point of design.

The link hardware transfers data between nodes. A central switch or cube topology is used for data routing. The link interface is implemented as an additional card which is connected to the bus of node computer. Optical link technology of today realizes the same transfer bandwidth as the internal bus bandwidth. The link assumes closed connection. Only small number of types of trusted nodes are connected. The communication between nodes is performed without checking access right and converting data format. A specially designed efficient and simple communication protocol is used for the communication. The standard LAN protocol such as TCP/IP is not used.

LAN interfaces are connected to one or multiple nodes. Clients access the cluster server by using a standard LAN protocol. The operating system of the server provides a single system image to clients. The operating system provides a cluster wide distribution transparent file system, distributed shared memory mechanism [5], inter-process communication mechanism and load balancing mechanism. These functions are not exported outside of the cluster because of efficiency and security.

2.3 Merits and Comparison

(1) Merits

The cluster server configuration combines existing hardware. The initial system cost is made low by combining mass product version of computers or combining computers already owned by the user. The developmental cost is also reduced because the development of a new core architecture is not required. The reduction shortens the system development time, which is important to catch up with the rapid progress of

[1]RT PC is a registered trademark of International Business Machines Corporation.

[2]UNIX is a registered trademark of AT & T Bell Lab. Inc.

[1]Transputer is a trademark of Inmos Corporation.

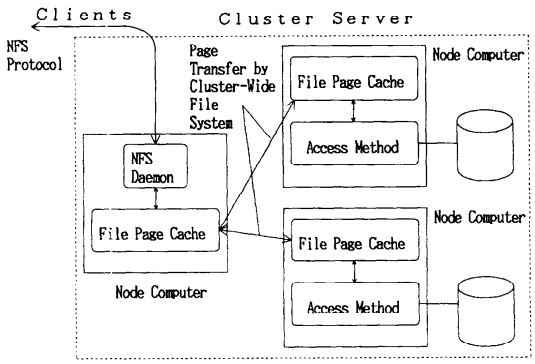

Fig. 3 The Network File Service by a Cluster Server.

semiconductor technology.

The cluster server provides a wide range of scalability. System configuration varies from a collection of a small number of old version machines to a collection of a large number of tightly coupled machines.

(2) Comparison to multiprocessor architecture

The tightly coupled architecture provides a desirable parallel processing mechanism of machines with around 10 processors. The performance difference between tightly coupled machines and cluster machines is not so clear for areas of high throughput job processing and file service. Not only CPU load but also I/O load are distributed by the cluster server architecture. The comparison depends on the inter-node communication cost of cluster machines.

When the number of processors in a system becomes greater than several tens, non-uniform memory access (NUMA) class architectures [3] and loosely coupled multiprocessor architectures [4] become standard. For example, a hierarchical bus system can connect such number of processors with non-uniform memory access time. The cluster server architecture connects a large number of processors by connecting a number of nodes or by connecting tightly coupled machines.

Multiprocessor users prefer implementing their programs based on the shared memory model, and this is a strong motivation of NUMA architecture. However, the distributed shared memory approach makes possible to provide the shared memory model by using loosely coupled machines. This technology eliminates the difference of programming model between NUMA multiprocessors and loosely coupled multiprocessors. They are compared directly by their cost and performance.

Operating systems of NUMA machines must copy frequently accessed pages to local memory to reduce access cost. The process creation and migration, which are needed to maintain load balance, disturb the access locality. The operating system reconstructs the process working set and file buffer pool by copying pages from node to node after creating or migrating processes. Operating systems of loosely coupled multiprocessor systems do the same thing by explicitly invoking communication link hardware. The total performance is determined by the cost of communication.

Current approaches to the operating system design of cluster servers are based on the LAN distributed operating system technology [6]. Improvement of them are possible to considerable extent. Concluding the comparison of architectures should be postponed until a specialized operating system for loosely coupled machines is implemented.

3. Reduction of Communication Cost

3.1 Communication Mechanism

3.1.1 Operation Modes of the Cluster Server

The measure of the communication cost depends on operation modes of the cluster server.

(1) File server mode

The cluster server operates as a server of network file system such as NFS. The service is simply realized by executing its server process on a node. Figure 3 shows an overview of the structure. The cluster-wide file system puts and gathers file pages on cluster nodes to/from the network file server process. The total performance is improved by parallel processing and a smart cache allocation. For example, different pages of a single file are placed on multiple nodes to perform parallel disk operation. On the other hand, a specific page of a file should be cached on one node and other nodes are better to prevent from placing the cache pages locally to use of the cache area efficiently. These strategies are only possible when the response of cluster-wide file system is sufficiently short.

(2) Job server mode

Job execution requests are submitted from command interpreters on client workstations to server processes on the cluster via LAN. These jobs are executed on an arbitrary node to balance the load of nodes. File system performance is important because they access files on arbitrary nodes. The CPU and memory consumption of the file system should be small to attain high throughput of job processing. The response of file access is not important because the access delay is canceled by multiprogramming.

(3) Parallel processing server mode

Parallel processing on the cluster server is based on the distributed shared memory mechanism and synchronization mechanism such as inter-process communication. A virtual space is shared among processes on different nodes by moving pages between nodes. The consistency maintenance mechanism determines the timing of page transfer by catching page faults. The execution time of the page and control data transfer becomes important for the parallel processing server operation.

3.1.2 Distributed File System [6]

It is important to investigate the access cost of an existing distributed file system to build an efficient cluster-wide file system. The NFS on UNIX system is an appropriate candidate because of its popularity.

The file system of UNIX is implemented by the cache management mechanism and access methods corresponding to media. The cache system contains file pages whose size is typically 8k byte. The "read" system call handler searches the cache first. An access method is invoked when the cache misses. The local disk file system and NFS are typical access methods.

The remote file read mechanism using NFS is composed of:—

1) Cache look up,
2) Free page allocation when cache misses,
3) Send request to server node by invoking network service,
4) Cache look up on server node,
5) Page acquisition,
6) Page transfer by network service,
7) Passing of the page to user.

Table 1 shows the total steps of client and server side programs. The figures are derived from the OSF/1 operating system code written in C language. The network service corresponds to 3) and 6), which include RPC, UDP, IP and Ethernet interface driver. It is apparent from the table that the principal part of the program is the network service layer.

The packet building is the main reason of the overhead of the layer. The typical file page size is 8k, but the Ethernet system can transfer packets of less than 1.5k byte. The network service layer needs to generate a series of packets of the size. The sender side divides a file page into multiple packets and associates protocol headers to them. The receiver side strips the headers and builds a file page from multiple packets. These are done by allocating, copying and freeing linked buffers called "mbuf." They increase not only program steps but also size of data copied by processor.

3.1.3 Remote Page Access

Distributed shared memory is implemented by a similar mechanism to the distributed file system. As described earlier, the grater part of the overhead is caused by the page transfer mechanism.

3.1.4 Inter-Process Communication

The performance requirement of the inter-process communication is tight because it is expected to be executed very frequently. The average data size of arguments of inter-process communication is known to be small [8], and the actual data transfer takes only small part of the overhead. A large part of the overhead is caused by the synchronization and packet marshaling mechanism. The synchronization mechanism includes process switching and waiting queue management. The packet marshaling cost refers to packing and unpacking a packet from/to complicated data structures of arguments of the inter-process communication.

Table 1 Static Steps of NFS.

Layer	Static Steps (C source code)
passing of page to user	114
cache lookup	25
free page allocation	50
page read from disk	493
network service	2591

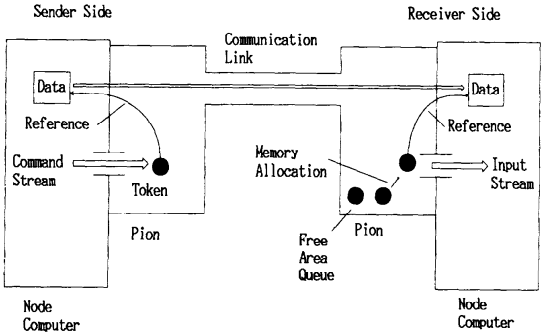

Fig. 4 An Overview of Communication System Using Pion.

3.2 CPI and Pion

We have designed an inter-node communication mechanism with an architectural support called Pion and a protocol layer called CPI (Communication Primitive Interface) to reduce overhead. CPI provides a common page transfer mechanism to access remote resources including file and memory, and provides an interprocess communication mechanism of RPC (remote procedure call) style.

3.2.1 Page Transfer Mechanism

(1) Overview

CPI accepts requests of putting pages to remote nodes and getting pages from remote nodes. The page is identified by virtual address which consists of virtual space identifier and offset within the space. The virtual space identifier identifies a file or logical address space.

Pion is attached to a node computer as shown in Fig. 4. Detailed specification of it is shown in (3). Pion of sender node generates a data stream, which is handled by inter-node communication link hardware, from data scattered on memory. Pion of receiver node allocates memory areas and receives data into them. Pion interacts with the processor via FIFO buffer.

Key ideas to reduce overhead are:

1) An architectural support for data stream generation and receiving,
2) A special kernel layer for page transfer.

(2) Remote file access sequence

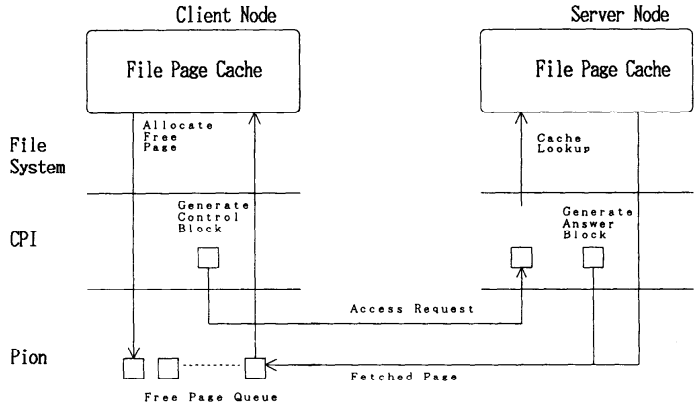

Fig. 5 File System Layers.

A remote file is accessed by the following sequence. Figure 5 shows the relationship between system layers. The file system layer allocates a free page from file page cache and asks CPI to copy remote page contents into it. CPI creates a control block to invoke the page fetch mechanism on remote node. CPI passes a token which holds the block address and size to Pion.

Pion on server side receives the control block and allocates a free memory area to it. Its address and size are passed to CPI in a form of token. CPI searches the file page cache based on the information stored in the block. When the requested page is found, CPI creates an answer block and passes two tokens designating the block and found page to Pion. The client side Pion allocates two areas to them, and passes two tokens to CPI. CPI analyzes the answer block and returns the address of received page held in the token to file system. The acquired file page is different from the one allocated by the file system at the time of request. The allocated page is pooled in Pion for future use.

When the CPI of server side can not find the page, it up-calls to the file system. CPI is notified after the page has been fetched from the disk.

(3) Pion design

Pion accepts series of command tokens each of which is the pair of address and size of a memory region to be transferred. Multiple tokens may correspond to a contiguous region on receiving node. The instruction for allocating memory on receiver node is kept in the command token. Pion associates the information to transmitting data. Pion keeps the free buffer area for data receiving, and the memory of required size is allocated without software intervention. The free memory area is kept by four queues of pointers each of which points to a free memory area. The four queues correspond to sizes of 16, 128, 4k and 8k bytes areas. When the memory area shortage occurs, Pion broadcasts discontinuation message to other nodes. The data transfer is initiated without hand-shaking in other case. The flow control scheme enables the connection-less transfer.

Software is freed from packet marshaling and de-marshaling by the Pion function. If the packet size of a communication system is fixed, software needs to build data structure by gathering multiple packets as seen in NFS implementation.

By contrast, the memory area of exactly the requested size is allocated and the data is directly moved into it by the Pion design.

The allocated buffer can be passed to the upper layer without copying or modifying. For example, when the requested data size is equal to the physical page frame size, it can be mapped into an user space immediately. When the data size is equal to file buffer page, the file system can exchange free page with received page by reference. Pion design enables to designate receive buffer size from sender node.

(4) A special kernel layer for page transfer

CPI implements the page transfer by the special protocol. An interrupt handler of CPI layer scans the page cache directly without invoking daemon process. The approach reduces overhead of process invocation and inter-process communication. Page transfer layer implemented by CPI is clearly distinguished from page allocation strategy implemented by upper layers, and CPI does not depend on a specific allocation strategy.

3.2.2 Inter-Process Communication

Pion provides the packet marshaling and buffer allocation service suited for a RPC style inter-process communication. An argument of the RPC is designated by three values; its address, size and pass direction of "in" and/or "out." Pion of caller side scans the argument list, and transfers the arguments marked "out." Pion of receiver side allocates a block to keep both "out" and "in," and receives "out" data. Callee process fills the "in" data. Pion of caller side copies back "in" data to the caller process address space by re-scanning argument list of caller.

A simple mechanism of CPI implements a RPC by

the service. The caller process pushes arguments on its stack and invokes a RPC primitive which is implemented by CPI. The CPI passes the stack pointer of caller process to Pion and waits for the answer. The callee side CPI builds a waiting queue which links blocks allocated by Pion. When a callee invokes the primitive of RPC receive, CPI simply dequeues a block and passes it to the callee.

4. Implementation

CFS (Cluster File System) is a file system using CPI. We have designed and coded CFS and CPI as an additional function of the OSF/1 operating system. Currently, the system is being debugged. The target hardware is a cluster of IBM PS/55 workstations. The workstation is equipped with the Intel 80386 CPU of 25M Hz clock, 16M byte memory and disk storage.

The communication link and Pion functions are emulated by another dedicated PS/55 system. The emulator and node machines are connected by the bus signal extender. The emulator can access physical memory of node machines by the hardware. Processors of sender nodes place Pion command tokens on their memories and generate interrupts to Pion emulator. The emulator tailors the memory image on receiver node, creates tokens on memory, and generates an interrupt on receiver.

CFS is integrated into OSF/1 kernel as a component of V-node file system. CFS provides a single directory tree among the cluster. The tree is built by executing mount operation on every node. User program can access file on any node transparently. The NFS protocol is used to attain cache coherence between nodes. The client side file cache is invalidated by time stamp. The file page is transferred by page transfer service of CPI. The control data, such as a file open request, is transferred by the inter-process communication mechanism.

An actual optical link system is scheduled to be implemented. A cross-bar switch of 8 edges switches the communication. The Pion function is going to be implemented by a processor and hard-wired logic on the interface card.

5. Discussion

CFS performance is able to be estimated based on the comparison of static steps of NFS and CFS. Two types of cost values are used for the estimation. The total cost value "T" corresponds to the access response and the CPU cost value "C" corresponds to CPU usage. The cost value "C" is equal to the sum of C language source code static steps of client and server. The unit of "C" corresponds to approximately 1 micro second. The value "T" is calculated by adding the latency of physical link and disk access to "C" value. A transfer of 10 byte through physical link is assumed to correspond to an unit cost value and the disk access cost is measured by its access time in micro second. Namely, "T" of remote disk access is gained by:

$$T = (\text{program steps of client}) + (\text{program steps of server}) + 0.1 * (\text{bytes of data transferred by communication link}) + (\text{disk access latency in micro seconds})$$

Disk access latency is assumed to be $2*10^4$ (20 m sec.) units.

Table 2 Remote File Access Cost of CFS and NFS.

Case	CPU Cost		Total Cost	
	NFS	CFS	NFS	CFS
Page Transfer Cost	5999	485	6818	1304
Local Cache Hit	1015	1015	1015	1015
Remote Cache Hit	7014	1500	7833	2319
Remote Cache Miss	7507	1993	27507	21993

Table 2 shows the costs of file page transfer and remote file access in three cases. The "total cost" column values refer to the cost of transfer of a 8k byte file page between file cache of two computers by using CPI and LAN network service. The CPU cost is reduced by 92% and the total cost is reduce by 81%.

The file access costs of NFS and CFS in the client cache hit case are same because they use the same mechanism. A page is transferred between nodes when the client side cache misses and the server side cache hits. The disk is accessed when all caches miss.

The actual file access time is affected by look ahead mechanism. In CFS, costs of local file access and transferring page between nodes are approximately same. It enables to fetch the next page from remote server node in parallel with reading current page. This affects cache allocation strategy. The local cache is not always necessary and it is sufficient to fetch from remote node for infrequently accessed files. It means that cluster-wide cache allocation scheduling improves performance.

6. Related Work

The improvement of the implementation of communication software, the introduction of architectural support and the design of light-weight protocol have been done to reduce communication cost.

The 4.3 BSD Reno system addresses the improvement of communication software efficiency [7]. Macro statements are introduced to operate "mbuf" structure efficiently. Remapping of virtual memory is used to pass mbuf data to link hardware. These improvements reduce CPU overhead by 12%.

Implementation of communication layers by hardware is also tried. Nectar [1] is a general purpose pro-

tocol engine implemented by a large scale hardware using RISC processor.

V system [2] provides VMTP protocol on IP for efficient implementation of RPC.

Our approach provides an efficient way of data transfer with assuming a relatively small scale architectural support and a specific data link layer protocol. It improves the data passing efficiency among file system, communication software and link hardware.

7. Conclusion

The progress of optical link technology brings a new era of the loosely coupled multiprocessor architecture. We have shown that a simple architectural support and a simple software mechanism provide a large amount of improvement in communication performance. The page transfer time is expected to be reduced over 80% by this approach in comparison to NFS. Our next target will be the showing of the validity of cluster server architecture through the evaluation of total system performance.

Acknowledgements

We wish to thank Dr. Kamimura for his invaluable suggestions and helpful discussions.

References

1. ARNOULD, E. A. et al. The Design of Nectar, A Network Backplane for Heterogeneous Multiprocessors, *the Proc. of Architectural Support for Programming Languages and Operating System* (1986), 205-216.
2. CHERITON, D. R. The V Distributed System, *Comm. ACM*, **31**, 3 (1988), 314-333.
3. GARCIA, A., FOSTER, D. and FREITAS, R. The Advanced Computing Environment Multiprocessor Workstation, *Research Report RC-14419, IBM Research Division* (1989).
4. GOODMAN, J. R. and WOEST, P. J. The Wisconsin Multicube, A New Large-Scale Cache-Coherent Multiprocessor. *Proc. 15th Annual International Symposium on Computer Architecture* (1988), 422-431.
5. LI, K. Shared Virtual Memory on Loosely Coupled Multiprocessors, Ph.D Thesis, Yale (1986).
6. LEVY, E. and SILBERSCHATZ, A. Distributed File Systems, Concepts and Examples, *ACM Comput. Surv.*, **22**, 4 (1991), 321-374.
7. MACKLEM, R. Lessons Learned Tuning the 4.3 BSD Reno Implementation of the NFS Protocol, *the Proc. of USENIX Winter '91* (1991), 53-64.
8. SCHROEDER, M. D. and BURROWS, M. Performance of Firefly RPC, *ACM Trans. Comput. Systm.*, **8**, 1 (1991), 1-17.
9. TANENBAUM, A. S. and RENESSE, R. V. Distributed Operating Systems, *ACM Comput. Surv.*, **17**, 4 (1985), 419-470.
10. THEIVER, M., LANTZ, K. and CHERITON, D. Preemptable Remote Execution Facilities for V-System, *Proc. of 10th ACM International Symposium on Operating System Principles*, **19**, 5 (1985), 2-12.

(Received May 15, 1991; revised November 26, 1991)