論文98-35C-6-2

HIPSS : SPAX(주전산기 IV) RAID 시스템

(HIPSS : A RAID System for SPAX)

李 尙 玟 *, 安 大 榮 *, 金 重 培 *, 金 鎭 杓 *, 李 海 東 *

(Sang-Min Lee, Dae-Young Ahn, Joong-Bae Kim, Jin-Pyo Kim, and Hae Dong Lee)

요 약

병렬 처리 시스템을 이용한 대용량 온라인 트랜잭션 처리(OLTP: on line transaction processing)와 같이 고 성능, 고 신뢰성을 요구하는 응용 환경에서 RAID 는 입출력 시스템에 필수적으로 요구되는 병렬 디스크 입출력 기술이다. 본 논문은 대용량 OLTP를 주 응용 분야로 설계된 주전산기 IV에 장착될 RAID 시스템인 HIPSS의 구조 및 구현 내용에 대하여 다루고 있다. HIPSS는 고 성능, 고 신뢰성, 외부 인터페이스의 표준화 및 모듈화, 편리한 시스템 관리 등을 설계 목표로 구현된 범용 RAID 시스템으로서, 10개의 독립적인 입출력 채널, 대용량의 데이터 캐쉬, 패리티 연산 하드웨어를 제공하여 시스템 성능 향상을 도모한다. 외부 정합 하드웨어를 쉽게 교체할 수 있게 설계하여 호스트 정합의 재구성이 용이하며, 또한 전원, 제어기의 이중화, 디스크 hot swapping 등의 기능을 제공하여 시스템의 신뢰성을 향상시킨다. HIPSS는 현재 구현이 완료되어 PC와 주전산기 IV를 이용한 기능 시험을 성공적으로 수행하였으며, 성능 개선 요소를 찾기 위한 시험을 수행 중에 있다. 본 논문에서는 HIPSS 시스템의 구조에 대한 자세한 설명과 구현 결과를 중심으로 기술한다.

Abstract

RAID technology that provides the disk I/O system with high performance and high availability is essential for OLTP server. This paper describes the design and implementation of the HIPSS RAID system that has been developed for the SPAX OLTP server. HIPSS has the following design objectives: high performance, high availability, standardization and modularization of external interface, and ease of maintenance. It guarantees high performance by providing 10 independent I/O channels, large data cache, and parity calculation engine. Hardware modularization of the host interface makes it easy to replace host interface hardware module. By providing dual power supply, dual array controller, and disk hot swapping, it provides the system with high availability Implementation of HIPSS and integration test on SPAX has been completed and performance measurement on HIPSS is now going on. In this paper, we provide the detail description for HIPSS system architecture and the implementation results.

I. 서 론

최근 병렬 처리 컴퓨터를 이용한 많은 응용 환경에서 입출력 성능이 시스템 성능의 주된 저하 요인으로 인식되면서 입출력 성능 개선을 위한 많은 연구들이 집중적으로 수행되었다. 이러한 노력의 결과 중에 하나로서 디스크 어레이 기술이 등장하게 되었으며, 1980년대에 RAID(redundant array of inexpensive disks)[1] 라는 이름의 기술로 발전되어 일반에게 널리 알려지게 되었다. RAID는 여러 개의 디스크들을 병렬로 운용하여 하나의 고성능 대용량 디스크처럼 사

* 正會員, 韓國電子通信研究院 컴퓨터構造研究室
(Computer Architecture Section, ETRI)

接受日字:1997年10月30日, 수정완료일:1998年5月25日

용할 수 있게 하는 입출력 기술로서, 입출력 요구를 여러 디스크에서 병렬(parallel) 혹은 동시(concurrent)에 처리하여 입출력 성능을 향상시킬 뿐 아니라, 보조 정보(redundancy information)를 이용한 데이터 복구 기능을 제공하여 시스템의 신뢰성을 높일 수 있는 디스크 입출력 기술이다[2] [3] [4]. 특히, 병렬 처리 시스템을 이용한 대용량 온라인 트랜잭션 처리와 같이 고 성능, 고 신뢰성을 요구하는 응용 환경에서 RAID 기술은 입출력 시스템에 필수적으로 요구되는 기술 요소들이다.

본 논문은 현재 한국 전자 통신 연구원에서 개발중인 SPAX(scalable parallel architecture computer based on X-bar network) 시스템의 입출력 장치로 개발된 HIPSS(high performance storage system) RAID 시스템의 구조와 구현 내용에 관하여 기술한 것이다. SPAX는 강력한 SMP(symmetric multi-processing) 구조의 컴퓨팅 노드들을 고성능 상호 연결 망으로 묶는 클러스터링(clustering) 구조의 고성능 병렬 처리 컴퓨터이다. 주 응용 분야는 OLTP와 DSS (decision support system)로서 시스템에 요구되는 입출력 요구들을 최대한 많은 입출력 장치들로 분산 시키고, 입출력 처리 경로 뿐 아니라 입출력 장치 자체에 대한 높은 신뢰성 제공을 요구하였다. HIPSS는 SPAX 시스템이 요구하는 높은 성능 및 신뢰성을 제공하면서도 외부 인터페이스의 표준화 및 모듈화, 편리한 시스템 구성 및 관리 등이 가능하도록 설계되었다. OLTP 환경에 적합한 RAID 5 구성에서의 성능을 극대화하기 위하여 RAID 5 성능의 저하 요인인 적은 쓰기 문제(small write problem)의 해결을 위한 하드웨어 및 소프트웨어 기능을 제공하도록 하였다.

그리고 높은 가용성을 제공하기 위한 신뢰성 보장을 위해 요구되는 디스크, 전원, 제어기 등에서 나타날 수 있는 특정 장애에 의해 발생하는 오류를 복구할 수 있는 기능과 시스템의 동작 중에도 오류가 발생한 디스크나 전원, 제어기를 교체할 수 기능을 지원하도록 하였다. 또한 호스트 정합은 다른 시스템에서도 쉽게 재구성하여 장착이 가능하도록 호스트 정합 하드웨어 모듈로 설계하였다. 이외에 시스템의 설치, 운용 및 관리를 위한 편리한 사용자 인터페이스를 제공하고, 성능 및 가용성에 대한 사용자의 요구에 따라 디스크와 제어기의 수 및 시스템의 구성을 용이하게 확장 및 변경이 가능하도록 하였다.

1. 관련 기술 연구

RAID 5 성능의 저하 요인인 적은 쓰기 문제는 데이터를 갱신할 때 패리티 갱신을 위해 이전 데이터 읽기, 이전 패리티 읽기, 새로운 데이터 쓰기, 그리고 새로운 패리티 쓰기 등 모두 4번의 디스크 입출력 연산을 수행함으로써 쓰기 응답 시간이 증가되어 쓰기 성능이 저하되는 문제이다. 이 문제를 해결하기 위하여 parity logging, floating parity, caching 등[5] 여러 기술들이 제안되고 있다. 이 기술들은 모두 여러 번의 적은 쓰기 요구들(small writes)을 모아서 한번에 큰 쓰기(large write)로 처리해 줌으로써 쓰기 오버헤드를 줄이는 것을 목적으로 한다. 이 중 caching 기술은 데이터와 패리티를 모두 디스크 캐쉬에 저장함으로써 모든 디스크 입출력 요구를 제거하여 입출력 성능을 개선시킬 수 있다. 여기서 쓰기 요구에 의해 메모리에만 저장되어 있는 데이터들은 주기적으로 디스크의 내용과 일치시킴으로써 데이터 손실을 방지한다. 그러나 전원 이상 등과 같은 예측 불가능한 사태가 발생하면 미쳐 디스크로 쓰지 못한 데이터를 손실할 수 있다. 이를 방지하기 위하여 대부분의 RAID 시스템들은 battery backup 기능을 갖는 비휘발성(non-volatile) 메모리를 이용하고 있다. IBM 사의 Hagar 시스템[10]이 그 한 예로써 특히 이 시스템의 경우에는 데이터 캐쉬와 패리티 캐쉬를 같은 메모리에 구현함으로써 패리티 연산을 위한 메모리 접근 시 발생할 수 있는 버스 혼잡(traffic)을 최소화한다.

성능 외에 높은 시스템 신뢰성(reliability)을 제공하기 위하여 많은 RAID 시스템들은 여러 가지 기능들을 제공하고 있다. 각 RAID마다 정의된 고장 감내 기능 외에 시스템 제어 장치 및 전원의 이중화, hot spare 디스크 제공, 소프트웨어적인 온라인 오류 복구(on-line failure recovery) 기능 등이 그것이다. Hewlett-Packard 사의 TickerTAIP 병렬 RAID 시스템[5] [8]의 경우를 보면 하나의 호스트 인터페이스를 갖는 여러 개의 제어기들을 두고, 이 제어기들을 내부적으로 고속의 망으로 연결함으로써 병렬 처리뿐 아니라 하나 이상의 제어기 오류를 감내할 수 있다. 또한 IBM의 Hagar시스템의 경우에도 하드웨어 장치들과 전원을 이중화함으로써 하나의 제어기 고장 시 다른 하나를 통해 시스템 수행이 가능하도록 한다.

2. 논문의 구성

본 논문은 다음과 같이 구성되어 있다. II장에서는 HIPSS 시스템의 구성 및 구현 내용에 대하여 설명하고, III 장에서는 성능 향상을 위하여 HIPSS에 구현된 캐쉬 방식에 대하여 상세히 기술한다. IV 장에서 결론으로 끝을 맺는다.

II. 시스템 구성

그림 1은 HIPSS 시스템의 블록 구조를 나타낸다. HIPSS는 두 개의 HIPSS 제어기와 여러 개의 디스크 스트링(Disk String)으로 구성된다. 하나의 HIPSS 제어기는 시스템 운용을 담당하며, 하드웨어를 구성하는 어레이 제어기(AP: array processor)와 어레이 제어기의 자원을 관리하여 입출력 요구를 처리하는 소프트웨어인 어레이 관리기(AM: array manager)로 구성된다. 각 어레이 제어기는 독립적인 호스트 인터페이스(fast/wide SCSI-II)를 통하여 외부 호스트에 연결될 수 있으며, 이중 경로를 둠으로써 두 개의 독립적인 병렬 디스크 시스템으로 운용되거나 하나의 디스크 접근 경로의 고장 시 다른 경로를 통한 데이터 접근이 가능하다. 또한, 어레이 제어기간 데이터 접근 경로(Inter-AP data path)를 통하여 하나의 AP 고장 시에 다른 제어기를 통해 고장 난 제어기의 데이터를 복구할 수 기능을 제공한다. 그리고 디스크 입출력 요구를 병렬로 처리할 수 있는 환경을 제공하기 위하여 모두 10 개의 독립적인 디스크 입출력 채널(fast SCSI-II)을 제공하며, 전체 채널을 두 개의 그룹으로 나누어 독립된 PCI 버스에 분산 배치함으로써 디스크와 데이터 저장 버퍼 사이에서 데이터 전송 시에 발생되는 부하를 분산시킨다.

여기서 PCI 버스는 프로그램 수행을 위한 프로세서 버스와 입출력을 위한 버스들을 분리시킴으로써 입출력 데이터의 전송 대역폭을 극대화하기 위하여 제공한다. 각 디스크 그룹은 5개의 디스크 입출력 채널과 각 채널마다 2 개의 디스크로 구성되며, 각 채널을 어레이 제어기간에 공유 시킴으로써 장치에 대한 접근 경로를 이중화한다. 또한 하드웨어적 요소들인 제어기, 전원 등도 이중화함으로써 시스템의 안정성을 향상시킨다.

그림 2는 HIPSS를 운용하는 하드웨어인 어레이 제어기의 구조를 나타낸다.

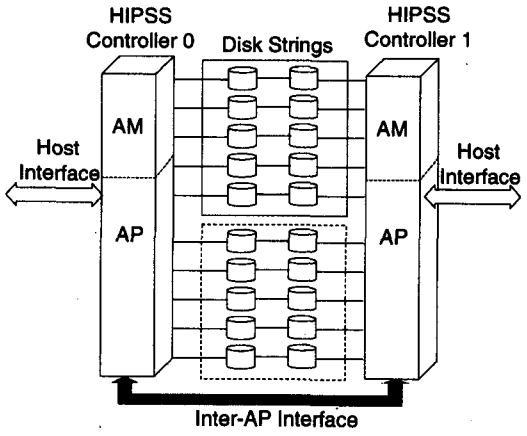


그림 1. HIPSS 시스템 블록도
Fig. 1. Block diagram of HIPSS system.

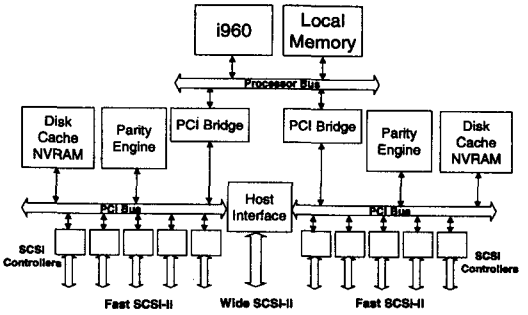


그림 2. 어레이 제어기 블록도
Fig. 2. Block diagram of array processor.

Intel i960 마이크로 프로세서와 2 MBytes 이상의 로컬 메모리, 데이터 블록을 저장하는 디스크 캐쉬 메모리, 패리티 연산을 수행하는 패리티 엔진과 PCI 버스 bridge, 그리고 호스트 및 디스크 인터페이스 제어기들로 구성된다. 여기서 디스크 캐쉬 메모리는 최소 16 MBytes에서 최대 128 MBytes까지 편리하게 구성할 수 있으며, EDC(error detection and correction) 기능과 전원 결함 등으로 인한 데이터 손실을 방지하기 위하여 battery back-up 기능을 제공한다. 패리티 엔진은 시스템이 RAID 5로 운용될 때 성능상 오버헤드가 되는 패리티 연산을 하드웨어적으로 고속으로 처리하는 장치로 패리티 데이터 캐쉬로부터 데이터를 전송하는 DMA 제어기(direct memory access controller)와 패리티 캐쉬로 이용되는 VRAM(video RAM), 그리고 하드웨어적으로exclusive-OR 연산을 수행하는 XOR 회로 등을 내장하고 있다. 이 장치는 디스크 캐쉬 메모리로부터 DMA 제어기에 의해 전송

된 데이터와 VRAM에 저장된 패리티를 내부의 XOR(eXclusive-OR) 회로에서 연산하고, 결과로 나온 새로운 패리티를 다시 VRAM에 저장하는 일을 수행한다.

그림 3은 4개의 기능 모듈로 구성되는 어레이 관리기의 구조이다. 어레이 관리기는 어레이 제어기의 하드웨어 자원을 관리하고, 운영 체제와의 정보 교환을 통해 디스크 입출력 요구를 처리하는 관리 소프트웨어로서 RAID 0, 1, 5, 1/0 등의 다중 RAID 지원, 디스크 데이터 및 패리티 캐싱 기능, 데이터 재생성(data reconstruction)과 디스크 구성 변경 등의 기능을 제공한다. 가장 상위의 메시지 인터페이스 모듈(MIF: message interface module)은 호스트로부터의 입출력 요구 메시지를 접수하고, 처리 결과를 다시 호스트로 전달하는 기능을 수행한다. 입출력 요구 메시지의 형태는 호스트 인터페이스로 SCSI 버스가 제공되는 경우 표준 SCSI 규약을 따르며, 우선 순위를 기반으로 여러 개의 입력 메시지 큐(IMQ: in-message queue)와 하나의 출력 메시지 큐(OMQ: out-message queue)를 관리함으로써 효율적으로 메시지를 처리한다. 다음으로 입출력 요구 스케줄링 모듈(JSCH: job scheduling module)은 입출력 처리 프로세스(service process)를 생성하여 요구된 입출력 명령을 처리하는 일을 수행한다. 입출력 응용 모듈(EMF: embedded function module)은 시스템의 형상 변경, 효율적인 데이터 및 패리티 배치, 디스크 캐쉬 및 패리티 캐쉬의 관리, 데이터의 재구성 등에 대한 정보를 제공한다. 이 모듈에서 시스템 형상 테이블(system configuration table)은 HIPSS에 여러 종류의 논리적 디스크 장치(LDU: logical disk unit)들이 존재하는 경우 이를 효율적으로 관리하기 위하여 제공된다. 하나의 LDU는 여러 개의 실제 디스크 장치(physical disk unit)들로 구성되며, 독립적인 디스크의 구성과 데이터 처리 방식을 갖는다. 시스템 형상 테이블의 각 엔트리는 운용하는 RAID 레벨의 종류, 분산 저장하는 디스크 수와 복사본 디스크의 수, 하나의 디스크에 저장되는 블록 수, LDU를 구성하는 실제 디스크 장치의 이름과 디스크 용량, 각 디스크 장치의 현재 상태, 그리고 spare 디스크 장치 번호 등의 정보를 저장한다. 모든 입출력 요구는 이 테이블의 검색을 선행하여 해당 RAID 레벨과 데이터 배치 함수가 결정되면, 요구되는 논리적 블록 주소를 물리적인 주소로 변환한 후 그 주소로 데이터 캐쉬를 검색하여 처리된다.

마지막으로 하드웨어 운용 모듈(HIF: hardware interface module) 모듈은 디스크 장치의 구동이나 메시지 송수신을 위하여 SCSI 제어기를 구동(SCSI control)하는 역할을 수행하며, 이외에 DMA 제어기 및 패리티 생성기(DMA & parity engine) 등을 포함한다.

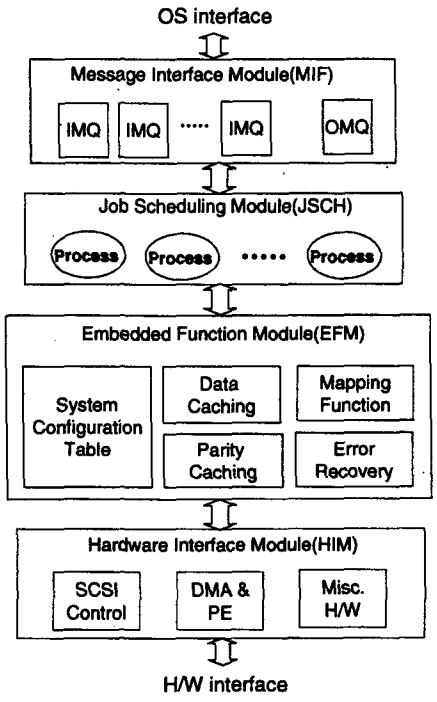


그림 3. 어레이 관리기 블록도
Fig. 3. Block diagram of array manager.

Ⅲ. 디스크 캐쉬 구현

본 절에서는 HIPSS에 구현된 데이터 및 패리티 캐쉬의 관리 구조와 운용 방식, 입출력 요구의 처리 과정과 오류 복구 기능, 블록 교체 방식 등에 대하여 상세히 기술한다.

1. 캐쉬 구조

그림 4는 데이터 및 패리티 캐쉬의 구조를 나타낸 것이다. HIPSS는 전체 입출력 채널을 두 개의 그룹으로 나누고, 각 그룹마다 디스크 캐쉬 메모리와 패리티 엔진을 두어 서로 독립적으로 이용할 수 있도록 한다. 그림 4는 각 그룹마다 RAID 5를 구성하는 LDU

가 존재하는 경우의 전체적인 캐쉬 구조를 나타낸다. 각 LDU가 이용하는 디스크 캐쉬 메모리 및 패리티 엔진의 정보는 로컬 메모리의 LDU 테이블에서 관리한다. LDU 테이블은 각 LDU의 캐쉬 관련 정보를 저장하는 테이블로서 각 엔트리는 LDU 구별자, 디스크 캐쉬 메모리 뱅크 구별자, 데이터 및 패리티 캐쉬 크기, 그리고 현재 캐쉬 상태 등의 정보를 저장한다. 이 테이블의 내용은 캐쉬 초기화 시 시스템 형상 테이블을 참조하여 작성된다.

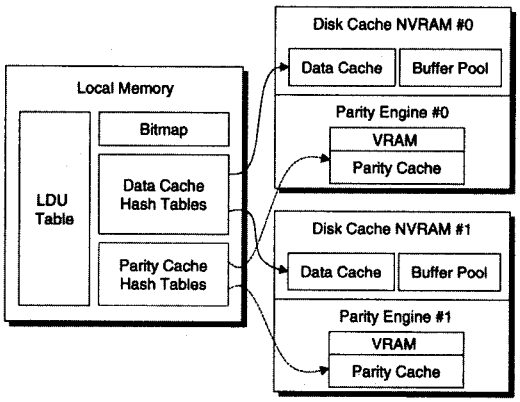


그림 4. 캐쉬 구조
Fig. 4. Cache structure.

해쉬표(hash table)는 입출력 명령에서 요구하는 블록의 캐쉬 내 위치 정보를 저장하는 테이블로서 모두 512개의 엔트리로 구성되며, 할당된 캐쉬 영역을 라인 단위로 나누어 관리한다. 캐쉬 라인은 하나 이상의 블록들로 구성되며, 라인 크기는 stripe element unit로 정의한다. Stripe element unit란 스트라입을 구성하는 디스크들 중 하나의 디스크에 저장되는 블록 수를 의미하며, 실제 디스크 상에 순차적으로 저장된다. 그리고 각 라인마다 두 개의 블록 주소를 할당한다. 하나는 이전 블록 주소(old block address)이고, 다른 하나는 새로운 블록 주소(new block address)로서 후자는 입출력 요구 시 처리되는 데이터를 저장하기 때문에 캐쉬 초기화 때 메모리가 할당되지만 전자는 쓰기 시 이전 데이터를 저장하는 용도로만 사용되므로 필요할 때마다 동적으로 할당하여 메모리 낭비를 줄인다. 라인을 구성하는 하나의 블록 크기는 효율적인 디스크 연산을 위하여 디스크 블록과 동일하게 512 bytes로 정의한다.

데이터 캐쉬는 읽기 및 쓰기 요구를 하나의 캐쉬에서 처리하며, 시스템에 여러 종류의 LDU들이 존재하는 경우 이를 지원할 수 있는 독립적인 캐쉬 관리 구조를 제공한다. 대용량의 비휘발성 디스크 캐쉬 메모리를 이용하기 때문에 성능 향상과 데이터 손실의 위험이 적다. 그리고 데이터 캐쉬에서 사용하는 블록들은 각 캐쉬 영역마다 별도로 관리하는 경우 발생할 수 있는 메모리 낭비를 방지하기 위하여 하나의 버퍼 풀로 관리한다. 패리티 캐쉬는 데이터 캐쉬와는 독립적으로 패리티 엔진 내 VRAM을 이용한다. 패리티 캐쉬를 데이터 캐쉬와 다른 메모리에 구현함으로써 동일한 메모리에 구현할 때 발생할 수 있는 데이터 캐쉬 성공률(cache hit ratio)과 성능에 미치는 영향을 줄일 수 있다. 또한 패리티 연산 시 데이터 전송으로 인한 버스 트래픽 증가는 패리티 엔진에 내장된 DMA 제어기를 이용하여 제거할 수 있다.

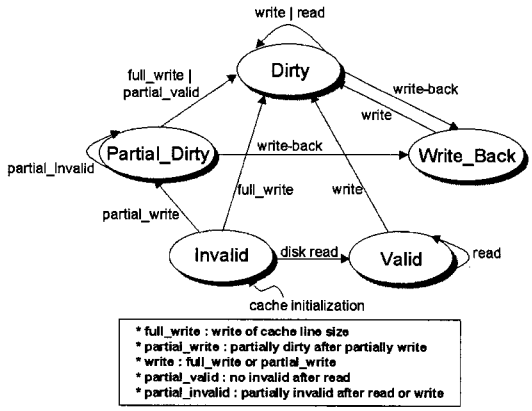
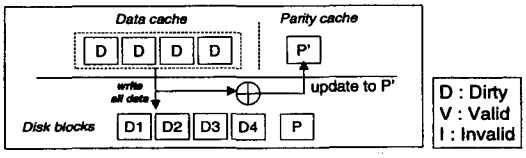


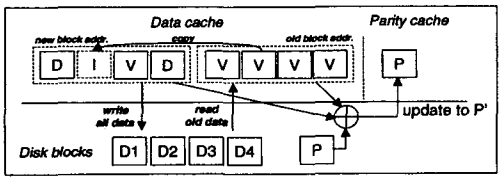
그림 5. 데이터 캐쉬 상태 천이도
Fig. 5. State machine of data cache.

그림 5는 데이터 캐쉬 라인 상태의 천이 과정을 나타낸 것으로 크게 Invalid, Valid, Dirty, Partial_Dirty, 그리고 Write_Back 등 다섯 가지의 상태를 정의한다. 각 상태는 캐쉬 라인을 구성하는 블록들의 상태를 반영하여 결정된 값으로 각 블록은 Invalid, Valid, 그리고 Dirty 상태 중 하나로 정의된다. 캐쉬 초기화 시에는 모든 블록들이 유효하지 않은 데이터를 저장하고 있으므로 Invalid 상태로 정의된다. 이 상태에서 호스트로부터 읽기 요구가 오면 요구된 블록 수에 상관없이 라인 전체를 디스크로부터 읽은 후 Valid 상태로 천이한다. 라인 전체를 읽는 것은 해당 블록만 읽으나 라인 전체를 읽으나 모두 한번의 디스크 접근으로 처리할 수 있기 때문에 응답 시간엔 별 차이가

없으며, 한번 요구된 블록이나 근접한 블록들이 다시 요구될 확률이 높기 때문이다. 그러나 읽기와 달리 쓰기의 경우에는 현재 라인 상태와 요구되는 데이터 크기에 따라 다음 상태가 결정된다. 라인 전체에 대한 쓰기(full_write) 또는 Valid 상태의 라인에 대한 부분적인 쓰기(partial_write)의 경우에는 Dirty 상태로 천이된다. 그러나 Invalid 상태의 라인에 대한 부분적인 쓰기의 경우에는 Partial_Dirty 상태로 천이한다. 결과적으로 Dirty 상태는 라인 내 모든 블록들이 Dirty 상태이거나 일부는 Dirty 상태이고, 나머지는 Valid 상태이지만, Partial_Dirty 상태는 일부는 Dirty 상태이고, 나머지는 Invalid 상태가 된다. 후자를 정의하는 이유는 매번 읽기 요구 때마다 라인의 전체 블록들의 상태를 검색해야 하는 오버헤드를 제거하기 위해서 이다. 즉, Partial_Dirty상태일 때는 각 블록에 대한 상태를 검색하여 Invalid 상태의 블록만 디스크로부터 읽어 오지만 Dirty 상태일 때는 바로 데이터를 전송한다. 시스템 이상으로 인해 메모리의 접근이 불가능할 경우 발생할 수 있는 데이터 손실을 방지하기 위하여 Dirty나 Partial_Dirty 상태 블록들은 주기적으로 디스크와 일치시켜야 하는데 이 과정이 끝나면 해당 라인은 Write_Back 상태로 천이한다. 이 상태는 비동기 연산으로 수행되는 write-back 과정과 수반되는 패리티 연산이 완료되었음을 의미하는 것으로 이 상태로 정의된 라인은 이후에 수행되는 패리티 연산에서 제외시킴으로써 연산 시간을 단축시킨다.



(a) Destage entire Parity Group : all blocks are dirty



(b) Destage Part of Group : some blocks are dirty

그림 6 4+1 RAID 5의 Destage 과정
Fig. 6 Destage process of 4+1 RAID 5.

2. 입출력 명령 처리

호스트로부터 입출력 요구가 오면 로컬 메모리의 LDU 테이블로부터 LDU를 결정한 후 해당 LDU가 이용하는 해쉬 테이블의 위치 정보로 요구한 블록의 캐싱 여부를 확인한다. 캐쉬 성공(cache hit)이면 바로 캐쉬 메모리에서 처리한다. 이 때 쓰기 요구에서는 새로운 블록 주소의 내용을 이전 블록 주소에 복사한 후 새로운 데이터의 쓰기를 수행한다. 그 이유는 현재 갱신된 데이터가 캐쉬로부터 축출될 때 수행되는 패리티 연산에서 이전 데이터를 위한 디스크 읽기 연산을 수행하지 않음으로써 연산 시간을 단축하기 위해서 이다. 그러나 캐쉬 실패(cache miss)일 때는 버퍼 풀로부터 빈 블록을 할당 받거나 모든 블록이 사용 중이라면 기존의 블록들을 할당하여 처리한다. 그런데 모든 입출력 요구가 하나의 캐쉬에서 처리되기 때문에 데이터 캐쉬에는 Valid 상태의 블록뿐 아니라 Dirty 상태의 블록들도 존재한다. 만일 다른 입출력 요구가 요구한 블록이 Dirty 상태이면 우선 해당 블록들을 디스크에 저장 해야 하는데 이를 destaging이라고 한다[10]. Destaging 과정으로 RAID 1이나 0/1은 복사본 디스크에 대한 쓰기를 병행해야 하고, RAID 5에서는 해당 패리티 블록을 갱신해야 한다.

그림 6은 5 개의 디스크로 구성되는 4+1 RAID 5에서의 destaging 과정을 나타낸 것이다.

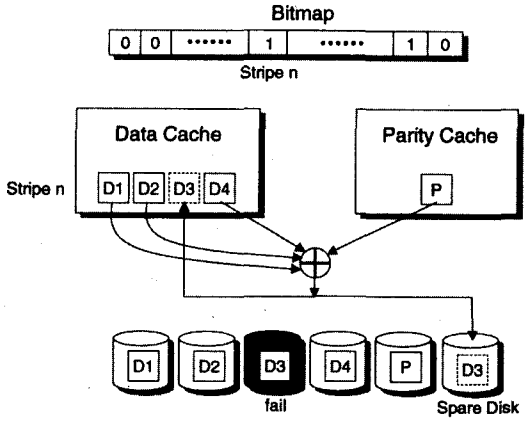


그림 7 RAID 5에서 데이터 재생성 과정
Fig. 7 Data reconstruction process of RAID 5.

그림 6의 (a)는 스트라입의 모든 데이터들이 캐쉬에 있고, 모두 Dirty 상태일 경우로 이전 데이터와 패리티를 읽는 디스크 연산이 필요 없이 Dirty 블록들을 exclusive-OR 함으로써 새로운 패리티를 산출할 수 있다. 그러나 (b)는 일부 블록들만 Dirty 상태일 경우로 라인 전체의 내용을 디스크로부터 읽어 이전 블록

주소에 저장한 후 패리티 연산을 수행한다. 새로운 데이터를 디스크에 쓸 때는 라인의 블록들 중 새롭게 캐쉬에 위치하는 블록들을 이전 블록 주소에서 새로운 블록 주소의 해당 블록으로 복사한 후 라인 전체를 한 번의 디스크 쓰기로 처리한다. 갱신된 패리티는 패리티 캐쉬에 저장되며, 데이터 캐쉬에서와 같이 다른 패리티에 의해 할당이 필요할 때 디스크로 저장된다.

그림 7은 데이터 재생성 과정을 나타낸다.

데이터 재생성 과정이란 디스크 읽기에서 읽고자 하는 데이터가 저장된 디스크가 고장났을 때 수행되는 오류 복구 과정을 말한다. RAID 1이나 0/1의 경우에는 복사본 디스크를 읽음으로써 바로 복구되며, RAID 5에서는 오류 데이터가 속하는 스트라입의 나머지 데이터들과 패리티의 exclusive-OR 연산으로 수행된다[3][9]. 그림 7은 RAID 5로부터 디스크 읽기를 수행할 때의 데이터 재생성 과정을 나타낸 것이다. 오류 데이터를 제외한 나머지 데이터나 패리티 블록 중 캐쉬에 없는 블록들은 디스크로부터 읽은 후 exclusive-OR 연산을 수행한다. 이렇게 복구된 데이터는 캐쉬와 동시에 여분의 디스크(spare disk)에 저장하고, 비트맵(Bitmap) 영역 중 해당 스트라입의 비트를 찾아 1로 셋팅한다. 그러나 디스크 쓰기의 경우에는 이 과정이 필요 없이 직접 데이터를 여분의 디스크로 쓰고, 비트맵의 해당 비트를 1로 셋팅하는 것으로 완료된다. 이 때 여분의 디스크는 고장난 디스크가 새로운 디스크로 교체되기 전까지 스트라입을 구성하는 디스크의 일원으로 동작한다. 따라서 디스크 고장이 발생하면 모든 입출력 요구는 비트맵의 검색부터 수행한다. 해당 비트가 1이면 이미 복구되어 여분의 디스크에 저장되었음을 의미하는 것으로 데이터 재생성 과정의 반복 수행 없이 여분의 디스크를 통해 처리한다. 이 방식은 데이터 복구로 인해 다른 입출력 명령들의 처리가 많은 시간 동안 지연되는 것을 방지할 수 있다.

3. 캐쉬 블록 교체

캐쉬 블록 교체는 새로운 데이터를 저장할 때 캐쉬에 빈 블록이 없어 기존에 저장되어 있는 블록들을 할당하는 과정이다. 일반적으로 Valid 상태 블록은 바로 할당되지만 Dirty 상태 블록의 경우에는 destaging 과정을 수행해야 한다. RAID 5는 destaging 과정 중에 패리티 갱신도 수행해야 한다. 따라서 자주 참조되는 데이터뿐 아니라 동일한 스트라입의 다른 블록들도 오랫동안 캐쉬에 유지하는 것이 무엇보다 중요하다. 이를 위해 HIPSS에서는 SLRU(segmented-LRU) 방식을 적용한다[7]. 일반적인 LRU 방식은 참조 시간으로 블록들을 관리하기 때문에 참조 빈도수가 낮은 데이터들에 의해 캐쉬가 점유됨으로써 캐쉬 성공률이 저하될 수 있는 문제점이 있다. 이에 반해 SLRU 방식은 빈도수를 기반으로 하는 LRU(frequency-based LRU) 방식이기 때문에 최근에 수행되었지만 다시 수행될 확률이 낮은 데이터들은 캐쉬로부터 빨리 축출되도록 하여 캐쉬 낭비는 줄이고, 캐쉬 성공률은 높일 수 있다.

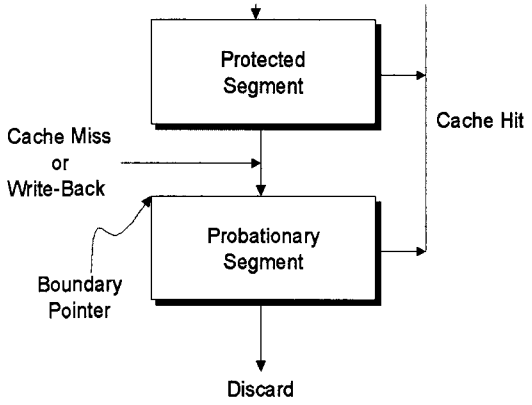


그림 8. SLRU 캐쉬 구조
Fig. 8. Structure of SLRU cache.

그림 8은 SLRU 방식을 적용한 캐쉬 구조로서 LRU 스택의 구조는 논리적으로 보호 세그먼트(protected segment)와 임시 세그먼트(probationary segment)로 나누어 관리된다. 각 세그먼트의 크기는 성능상 중요한 변수로서 일반적으로 보호 세그먼트의 크기를 두 배 이상 크게 할당한다. 보호 세그먼트란 캐쉬에 저장된 후 한번 이상 참조된 블록들을 유지하는 영역으로 캐쉬 성공된 블록들은 이 세그먼트의 가장 상위 블록으로 이동한다. 항상 캐쉬 검색은 보호 세그먼트의 가장 상위 블록부터 수행되기 때문에 자주 참조되는 블록을 검색하는데 걸리는 시간을 상당히 단축시킬 수 있다. 그러나 보호 세그먼트의 블록들 중 오랜 동안 참조되지 않는 블록은 임시 세그먼트의 가장 상위 블록으로 이동하여 관리된다. 임시 세그먼트란 새롭게 캐쉬에 저장되거나 오랫동안 참조되지 않아 보호 세그먼트에서 밀려난 블록, 디스크로 write-

back된 블록들을 유지하는 영역이다. 일단 임시 세그먼트에 저장된 블록이 다시 요구되면 보호 세그먼트의 가장 상위 블록으로 이동하고, 이 세그먼트 내에서 오랫동안 참조되지 않으면 캐쉬로부터 축출된다. 두 세그먼트는 하나의 이중 연결 구조로 관리되며, 두 세그먼트는 임시 세그먼트의 가장 상위 블록을 가리키는 경계 포인터(boundary pointer)로 구분된다.

4. 구현 결과

그림 9는 구현된 HIPSS의 겉 모습을 보여 준다. HIPSS는 크게 AP 보드, 디스크 모듈, 패키지 등 3개의 모듈로 구성되며, 그림 10과 그림 11은 각각 AP와 디스크 모듈의 구현 결과이다.

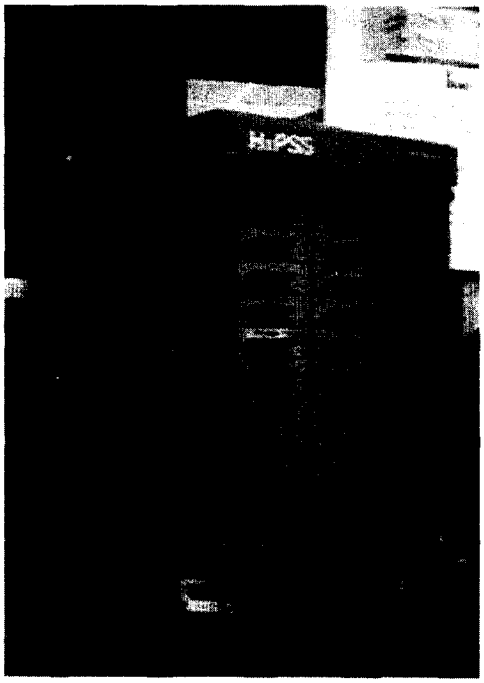


그림 9. HIPSS 시스템
Fig. 9. HIPSS system.

AP 보드는 12층의 PCB 기판 위에 구현되었으며, 30cm x 45cm의 보드 크기를 가진다. 프로세서는 인텔의 i960HD 33 MHz를 사용하였으며, 4MBytes의 로컬 메모리는 SRAM 모듈을 사용하였다. AM 프로그램은 AP 보드의 플래쉬 메모리에 저장되어 있으며, 보드 리셋 시에 로컬 메모리로 복사된 후 수행된다. 디스크 캐쉬 메모리는 ECC를 지원하기 위하여 16 MBytes 40-bit DRAM 모듈을 사용하도록 하였으며, 최대 128 MBytes까지 확장할 수 있다. 디스크 캐쉬 메모리의 PCI 버스 인터페이스 기능과 DRAM의 제어 회로 및 패리티 엔진은 Altera EPLD를 사용하여 구현하였다. 그리고 호스트와 디스크 인터페이스용의 SCSI 제어기는 Symbios사의 SYM53C875와 SYM 53C860을 사용하였다. HIPSS는 그림 9와 같은 패키지에 최대 2장의 AP보드가 장착될 수 있다.

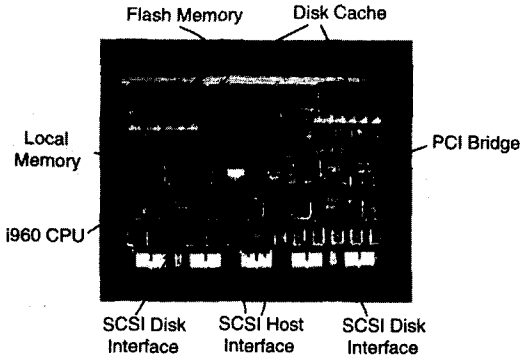


그림 10. AP 보드
Fig. 10. AP board.

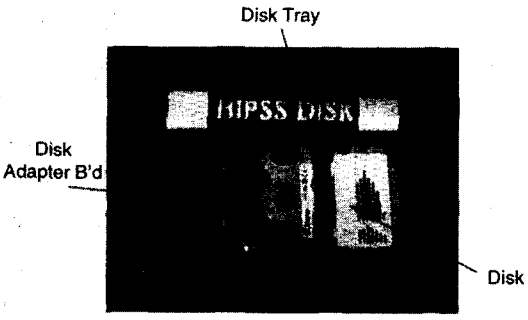


그림 11. 디스크 트레이
Fig. 11. Disk tray.

그림 10은 AP 보드의 구현 모습을 나타낸다. 그림 11은 디스크 모듈의 구현 모습으로서 디스크, 디스크 트레이(disk tray), 디스크 접속 보드(disk adapter board) 등 3 개의 서브모듈로 구성된다. 디스크 트레이는 하나의 디스크와 디스크 접속 보드를 장착하여 HIPSS 패키지에 쉽게 장탈착 할 수 있도록 지원하는 기구물이다. 디스크 접속 보드에는 live insertion을 지원할 수 있는 보조 회로와 콘넥터가 구현되어 있어서 전원이 켜진 상태에서 다른 기능 모듈에 영향을 주지 않고 디스크 모듈의 장탈착이 가능하다. 패키징 모듈에는 이중화된 전원 장치와 배터리, 냉각 장치 그리고 디스크 모듈과 AP 보드를 장착할 수 있는 SCSI

버스 백플레인으로 구성되어 있다.

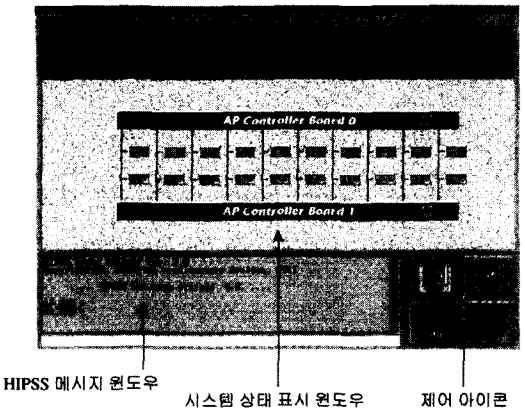


그림 12. HIPSS GUI
Fig. 12. HIPSS GUI.

그림 12는 HIPSS의 상태 및 구성을 제어하기 위한 그래픽 사용자 인터페이스이다. 사용자 인터페이스는 SUN 워크스테이션의 X window 환경에서 motif를 이용하여 구현하였으며, SUN 시스템과 HIPSS는 RS232C 규격의 serial line을 통하여 데이터를 전송한다. 화면의 구성은 시스템의 상태를 표시하는 창, HIPSS의 어레이 관리기로부터 전송되는 문자 정보를 보여주는 창, 그리고 시스템 제어를 위한 제어 아이콘들로 구성된다.

Ⅳ. 결 론

본 논문에서는 OLTP와 DSS를 주 응용 분야로 하는 병렬 처리 시스템인 SPAX의 성능 및 신뢰성 요구 사양을 만족시키기 위하여 개발된 HIPSS RAID 시스템의 구조와 구현 내용에 관하여 기술하였다. 개발된 HIPSS의 구조적인 특징은 첫째, 외부 정합 하드웨어 모듈을 쉽게 분리, 대체 가능하게 설계되어 SCSI 뿐 아니라 FC(fiber channel), ethernet 등의 망에 접속되는 환경에서도 쉽게 재구성할 수 있다는 것이다. 둘째로, 최대 128 MBytes의 디스크 캐쉬와 VRAM을 이용한 패리티 연산 하드웨어 그리고 패리티 캐쉬를 이용하여 RAID 5의 쓰기 성능 향상을 도모하였고, 셋째로 10개의 입출력 채널을 2개의 PCI 버스를 이용하여 2개의 그룹으로 나눔으로써 디스크와 캐쉬 간의 데이터 전송 시에 발생하는 부하를 분산 시켰으며, 또한 프로세서 버스(i960 버스)와 데이터 전송 버스(PCI 버스)를 분리하여 프로그램 수행 부하와 데이터 전송 부하를 분리한 것이다. 넷째로는 각각의 입출력 채널을 두 개의 AP가 공유할 수 있게 함으로써, 독립적으로 접근되는 두개의 논리적인 디스크로 사용되거나, 하나의 AP로부터 논리 디스크의 접근이 불가능할 때 다른 AP를 통하여 접근할 수 있는 기능을 가지도록 하였다.

HIPSS는 RAID 5의 성능 저하 요인인 적은 쓰기 문제를 해결하기 위하여 데이터 및 패리티 캐쉬를 구현하였다. 데이터 캐쉬는 읽기 및 쓰기 요구를 하나의 캐쉬에서 처리하며, 시스템에 여러 종류의 LDU들이 존재하는 경우 이를 지원할 수 있는 독립적인 캐쉬 관리 구조를 제공하고 있다. 데이터 캐쉬는 비휘발성 메모리로 구현하여 시스템 이상 등으로 인한 데이터 손실을 방지하고, 하나의 디스크 블록에 대해 두 개의 블록 주소를 할당함으로써 패리티 연산 시간을 단축시킨다. HIPSS에서 쓰기 요구는 직접 디스크로 가지 않고 메모리에 저장함으로써 완료되고, 주기적으로나 다른 입출력 요구에게 블록을 할당할 때마다 갱신된 블록들을 디스크와 일치시킨다. 이 때 RAID 5에서는 해당 패리티의 갱신을 위한 패리티 연산을 수행한다. 패리티 연산은 전담 하드웨어로 제공되는 패리티 엔진과 패리티 엔진 내 VRAM을 패리티 캐쉬로 이용하여 고속으로 처리한다. 이 때 패리티 연산을 수행할 데이터의 전송으로 발생하는 버스 트래픽은 DMA 제어기를 이용하여 제거될 수 있다. HIPSS는 블록 교체 방법으로 SLRU 방식을 적용하였다. 이 방식은 빈도수를 기반으로 하는 LRU 방식으로 캐쉬 영역을 논리적으로 보호 세그먼트와 임시 세그먼트로 분리하여 관리한다. 보호 세그먼트는 한번 이상 참조된 블록들이 위치하는 영역이고, 임시 세그먼트는 오랫동안 참조되지 않거나 새롭게 참조된 블록들이 위치하는 영역으로 블록 교체는 임시 세그먼트의 가장 하위 블록으로부터 수행된다. 시스템의 신뢰성 보장을 위해 HIPSS는 디스크 고장 시 오류 디스크에 저장된 데이터를 복구하는 데이터 재생성 과정을 일반적인 디스크 입출력 명령과 병행하여 처리한다. 이렇게 병행하여 처리함으로써 오류 복구 과정으로 인한 다른 입출력 명령들의 응답 시간 지연과 그로 인한 성능 저하를 줄일 수 있다.

현재 HIPSS는 기능 구현이 완료되어 PC와 SPAX에 장착하여 기능 시험을 종료한 상태에 있으며, 성능 개선 요소 도출을 위한 시스템 성능 특성 추출 실험을

수행 중에 있다. 앞으로, 실험을 통해 도출된 성능 보완 요소에 대하여 성능 개선 활동이 계속될 예정이며, 또한, 실시간 운영체제를 기반으로 한 AM을 구현하여 실시간 입출력 처리 기능을 제공함으로써, HIPSS를 멀티미디어 저장 시스템으로 확장할 계획이다.

참 고 문 헌

[1] D.A.Patterson, J.L.Hennessy, and R.H. Katz, A case for redundant arrays of inexpensive disks(RAID), *International Conference of Management of Data(SIGMOD)*, pp. 109-116, 1988.

[2] E.K.Lee, and R.H.Katz, Performance Consequences for Parity Placement in Disk Arrays, *International Conference on Architectural Support for Programming Languages and Operating systems*, pp. 109-199, 1991.

[3] M.Hollang, G.Gibson, and D.Siewiorek, Fast, on-line Failure Recovery in Redundant Disk Arrays, *Proceedings of the Fault Tolerant Computing*, 1993.

[4] K.Salem, Disk striping, *Proceedings of IEEE Data Engineering Conference*, LA, CA, pp. 336-342, Feb. 1986,

[5] P.M.Chen, et al, RAID : High-Performance, Reliable Secondary Storage, *ACM Computing Surveys*, vol. 26, no. 2, pp. 145-185, June. 1994.

[6] John Ousterhout and Fred Douglis, Beating the I/O bottleneck : a case for log-structured file systems, *Operating Systems Review*, 23(1), pp. 11-27, Jan. 1989.

[7] Ramakrishna Karedla, J.Spencer Love, Bradley G.Wherry, Caching Strategies to Improve Disk System Performance, *IEEE Computer*, vol. 27, no. 3, pp. 38-46, March 1994.

[8] Pei Cao, Swee Boon Lim, Shivakumar Venkataraman, and John Wilkes, The TickerTAIP parallel RAID architecture, *ACM Transactions on Computer-Systems*, 12(3), pp. 236-269, August 1994.

[9] R.R.Munts and J.C.S.Lui, Performance analysis of disk arrays under failure, *In Proc. 16th VLDB*, Brisbane, Australia, August 1990.

[10] Jai Menon and Jim Cortney, The Architecture of a Fault-Tolerant Cache RAID Controller, Proc. of the 20th Annual International Symposium on Computer Architecture, pp. 76-86, May 19.

저 자 소 개

李 尙 玫(正會員)

1968년 10월 6일생. 1991년 인하대학교 전자계산학과 졸업. 1991년 ~ 현재 한국전자통신연구원 컴퓨터구조연구실 연구원. 주관심분야는 병렬처리, 병렬 I/O 시스템

金 重 培(正會員)

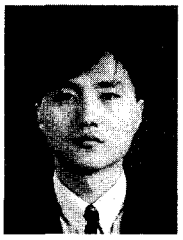
1960년 11월 23일생. 1989년 한남대학교 공과대학 전자계산공학과 졸업. 1989년 ~ 현재 한국전자통신연구원 선임연구원. 주관심분야는 병렬컴퓨터시스템, 병렬입출력, 영상처리

李 海 東(正會員)

1967년 2월 2일생. 1990년 서강대학교 전자공학과 졸업. 1992년 서강대학교 전자공학과(공학석사). 1996년 서강대학교 전자공학과(공학박사). 1996년 ~ 현재 한국전자통신연구원 선임연구원. 주관심분야는 CAD 시스템, 병렬처리 컴퓨터 시스템

安 大 榮(正會員)

1962년 2월 20일생. 1984년 한양대학교 전자공학과 졸업. 1986년 한국과학기술원 전기 및 전자공학과(공학석사). 1992년 ~ 현재 한국과학기술원 전기 및 전자 공학과 박사과정. 현재 한국전자통신연구원 컴퓨터구조연구실 선임연구원. 주관심분야는 병렬처리컴퓨터 구조, 병렬 I/O 시스템, 시스템 성능분석



金 鎭 杓(正會員)

1969년 5월 31일생. 1993년 연세대학교 전산학과 졸업. 1995년 연세대학교 전산과학과(이학석사). 1995년 ~ 현재 한국전자통신연구원 컴퓨터구조연구실 연구원. 주관심분야는 병렬처리, 컴퓨터 구조, networked I/O