□특 집□

실시간 시스템의 하드웨어 아키텍쳐

한국과학기술원 전산학과 컴퓨터구조 연구실 **윤현주 · 윤현수***

목 차

Ⅰ. 서 론
Ⅱ. 실시간 처리기의 요건
2.1 정확성
2.2 시간엄수성
2.3 적응성
Ⅲ. 사례 시스템 연구
3.1 SMART
3.2 적응성을 고려한 모듈 접근 방식
Ⅳ. 설계시의 주요 논점
Ⅴ. 결 론

Ⅰ. 서 론

실시간 시스템(real-time system)이란 결과가 나오는 시점이 중요한 시스템 혹은 외부에서 들어오는 입력에 대해 제한된 시간내에 반응하는 정보처리 시스템이다[12]. 이러한 시스템의 예로는 공정제어 시스템, 군사무기 제어, 전전자 교환기 시스템 등을 들 수 있다. 이 중 특히, 핵 발전소의 제어나 비행 제어와 같이 시간엄수에 대한 조건이 엄격한 시스템을 엄격한 실시간 시스템(hard real-time system)이라 하며, 이들의 제어가 잘못된 경우에는 시스템 자체뿐만 아니라 그것이 제어하는 다른 시스템 및 인간에게까지 심각한 영향을 미치게 된다.

이러한 실시간 시스템이 만족해야 할 중요한 요구 조건으로 정확성(correctness), 시간 엄수성(timeliness), 적응성(adaptability) 등을 들 수 있다. 실행 결과가 정확해야 한다는 것은 모든 시스템의 일반적인 요구 사항이며, 실시간 시스템의 경우는 그와 더불어 실행의 완료가 정해진 시간 조건을 만족해야 한다는 특수성을 가진다. 따라서, 실시간 시스템 아키텍쳐의 설계에 있어 이들을 고려해야 함은 필수적이다. 또 하나의 요구 사항인 적응성은 최근에 있어서의 기술 발전의 추세와 실시간 시스템이 적용될 수 있는 응용분야의 확대에 기인한 것이다. 첫째, 시스템을 구성하는 중앙 처리 장치, 입출력 장치, 기억 장치 등이 빠른 속도로 발전하고 있기 때문에 오랜 기간을 두고 사용해야 하는 시스템의 경우, 이러한 발전된 장치를 그때 그때 수용하여 이용할 수 있는 적응성이 필요하다. 둘째, 시스템이 이용되는 환경 자체가 동적이어서 그 변화에 잘 적응할 수 있는 하드웨어와 소프트웨어 구조를 가져야 한다. 세째, 실시간 시스템이 주로 큰 시스템의 부속 시스템(embedded dsystem)으로 들어가기 때문에 특수 목적에 적합하도록 설계되는 경우가 많으나 개발 비용 등을 고려해 볼 때, 약간의 노력으로 여러 시스템에 두루 적응될 수 있으면 좋을 것이다.

본 연구에서는 이상과 같은 요구 조건을 만족시키기 위해 실시간 시스템의 하드웨어 아키텍쳐가 고려해야 할 요건들을 살펴보고 사례 시스템들을 조사하였다. 먼저, 정확성과 시간 엄수성을 만족하기 위한 요건으로 예측가능성(predictability), 신뢰성(reliability), 고

* 종신회원

성능(high performance)의 세 가지를 제시하고 그의 중요성에 대해서 살펴 보았다. 특히, 시간 엄수성을 위해서 강력하게 요구되지만 성능이나 비용 등의 일반적 조건들과 상충되는 면이 많은 예측가능성에 대해 중점적으로 고찰하여, 하드웨어 아키텍쳐의 각 부분에서 고려되는 예측가능성을 살펴보고 사례 시스템으로 SMART 캐쉬[6,7]를 조사하였다. 그리고, 적응성을 구성요소 적응성(component adaptability)과 시스템 적응성(system adaptability)으로 나누어 생각해 보고, 하드웨어 이키텍쳐에서 특히 중요하게 고려해야 할 구성요소 적응성에 대해 고찰하였다.

본 논문의 구성은 다음과 같다. 2장에서는 실시간 처리기의 요건에 대해 살펴보고 특히 예측 가능성과 적응성에 대해 상술하였다. 3장에서는 2장에서 논의된 요건들에 중점을 두고 설계된 사례 시스템들을 살펴보았다. 그리고, 4장에서는 실시간 시스템을 실제로 설계할 때의 주요 논점에 대해 기술하고, 마지막으로 5장의 결론으로 끝을 맺었다.

II. 실시간 처리기의 요건

실시간 시스템은 그 실행 결과가 정확해야 한다는 일반적인 요구 사항 외에도 정해진 시간을 지켜 수행되어야 한다는 특성을 지닌다. 또한, 하드웨어 발전의 속도가 빨라지고 실시간 시스템을 이용하는 응용분야가 늘어나고 있는 환경로 적응성이 새로이 요구되고 있다. 이들은 하드웨어 설계뿐 아니라 소프트웨어와 응용 프로그램 등 실시간 시스템 전체를 구성하는 데 중점적으로 고려해야 할 항목들이다. 2장에서는 이 세 가지 요구사항이 각각 어떤 의미를 가지며 그를 고려한 하드웨어 아키텍쳐는 어떻게 되어야 할 것인가에 대해 살펴보았다.

2.1 정확성

일반적으로 실시간 시스템은 다른 큰 시스템에 포함된 하나의 부속 시스템으로 들어가서 중요한 제어를 담당하는 경우가 많다. 특히, 핵 발전소나 비행과 같이, 오동작이 일어나는 경우 큰 위험이 초래되는 응용분야에 이용되는 실시간 시스템을 엄격(hard)하다고 하는데, 엄격한 실시간 시스템의 경우는 보다 그 결과가 정확해야 하고, 시간 조건이 있을 경우 그를 엄수해야만 한다.

계산 결과가 정확하게 나와야 함은 계산의 자료를 얻는 입력부, 계산을 수행하는 연산부, 결과를 내는 출력부 등의 하드웨어와 소프트웨어가 안정되고 신뢰성이 있어야 함을 뜻한다. 컴퓨터 시스템에서 신뢰성(reliability)을 중점적으로 고려하는 것은 주로 고장 허용(fault tolerant) 구조를 연구하는 분야이다.

고장 허용성은 첫째, 고장의 발생을 미리 방지하는 고장 회피(fault avoidance) 기법, 둘째, 고장 발생시 그를 감지하고 발생 위치를 찾아내며 다른 곳으로 번지는 것을 막는 고장 차단(fault masking), 세째로 고장으로 인해 오류가 발생한 경우라도 그를 적절히 복구하거나 그 오류가 시스템에 영향을 미치지 않도록 하는 고장 감내(fault tolerance)의 기법들로 구현된다. 어느 단계에서 고장 허용성을 구현할 것인가는 응용분야의 특성에 따라 결정되며, 그를 하드웨어로 구현할 것인가, 소프트웨어로 구현할 것인가는 각각에 소요되는 비용, 요구되는 성능, 유연성(flexibility)의 정도에 의해 결정된다.

고장 허용성은 실시간 응용뿐만 아니라 여러 시스템에서 광범위하게 요구되는 특성이기 때문에 고장 허용 시스템에 대한 연구가 이미 많이 진행되었으며, 실제 사용하는 시스템들도 많이 개발되어 있다[5]. 그러나, 실시간 특성과 고장 허용의 특성을 동시에 중요하게 고려한 연구는 별로 없는 상태이며, 그 예도 찾아보기 어렵다. 실시간 시스템에서 대부분 고장 허용성을 고려하고 있긴 하지만, 연구의 초점은 고장 허용성이나 실시간성 중의 하나뿐인 경우가 많기 때문에 그 두 가지 특성의 상관 관계에 대해 연구된 결과는 보기 어렵다. 실시간 공정 제어 시스템에서 처리기 풀(pool)의 개념을 사용하여 고장에 대한 실시간 동적 대처로 실시간성을 구현하고자 한 Poolpo시스템[8]과 같은 접근 방식의 예를 볼 수 있다.

2.2 시간 엄수성

시간의 엄수는 실시간 시스템의 가장 중요한 요건이다. 이를 위해 고려해야 할 사항으로 예측 가능성, 신뢰성, 고성능 등을 들 수 있다. 첫째, 처리기의 성능이나 실행 동작을 사전에 미리 파악할 수 있는가의 여부에 따라 어떤 작업의 처리 시간 계산 정확도가 좌우되고, 이는 또한 한계 시간 제약을 가지는 많은 직업들이 어떻게 스케쥴링되어야 할 것인가에 큰 영향을 미치게 된다. 즉, 시스템이 얼마나 예측 가능한가 하는 것은 시간이 가장 큰 제약 조건인 실시간 시스템에서 아주 중요한 요구 사항이 된다. 둘째, 일반적

으로 실시간 시스템은 다른 큰 시스템에 포함된 하나의 구성 요소로 들어가고, 특히. 엄격한 실시간 처리를 요구하는 응용 분야들은 앞에서 언급한 바와 같이 시스템의 오동작을 허용하지 않는 경우가 대부분이다. 그러므로, 시스템의 하드웨어와 소프트웨어의 높은 신뢰성을 보장할 수 있어야 한다. 세째, 고속으로 수행할 수 있는 고성능 처리기는 작업 실행 시간을 단축시켜 한계 시간의 엄수와 보다 많은 작업의 스케쥴링을 가능하게 할 수 있다.

신뢰성에 대해서는 앞절에서 다룬 바와 같으므로 예측가능성과 고성능에 대해 좀 더 상세하게 보기로 한다.

2.2.1 예측가능성

엄격한 실시간 시스템은 시간을 엄수하는 것이 아주 중요한 요구 사항중의 하나이기 때문에 수행해야 할 작업의 정확한 수행 시간을 예측할 수 있어야 하는 것이 필수적이다. 수행 시간을 예측하기 위해서는 먼저 작업의 양이 정확하게 분석이 되어야 하는데 이는 컴파일러에 의해 이루어져야 한다. 컴파일러는 작업량의 계산뿐만 아니라 스케쥴링이나 운영 체제에서 필요한 정보도 추출해 낼 수 있어야 한다. 작업량이 분석된 후에는 그 작업에 걸리는 시간을 계산해야 한다. 이는 어떤 특정 명령을 수행하는데 필요한 사이클 수, 기억 장치 접근 시간, 명령 및 데이타 전달 시간, 다른 프로세스와의 통신이 필요한 경우 그 통신 시간 등 실제 하드웨어 상에서의 수행 시간에 따라 계산된다. 즉, 그러한 하드웨어 요소들의 실행 시간이 완전히 예측 가능해야 한다는 것을 의미한다.

Stankovic은 이러한 하드웨어 예측성을 구현하기 위한 방법으로 분할 원칙(principles of segmentation)을 제시하였다[12]. 분할이란 시스템의 여러 자원들을 요구되는 응용과 자원의 양에 맞추어 적절한 단위로 나누는 것을 의미한다. 그리고, 이러한 단위는 될 수 있는데로 고정된 크기를 가짐으로써 그 행동 양태를 쉽게 추산할 수 있도록 한다.

엄격한 실시간 시스템에서는 시간이 가장 중요한 자원이 되고 이를 어떻게 분할하는 것인가가 문제이다. 시간은 CPU 시간, 연결망 전송 시간, 내부 버스 사용 시간, 디스크 제어기 등을 사용하는 시간 등 여러가지 다른 성격의 시간들로 구성되어 있다. 이들은 각각 그 크기 정도(granularity)가 다르고 인터페이스하는 방법도 다르다. 작업의 수행예상시간이 예측가능하고 분할 단위의 크기들이 고정되면 스케쥴링 시에 할당해야 할 자원과 시간의 크기가 고정되기 때문에 임의의 대기 상태를 방지할 수 있어 한계 시간을 지키는 스케쥴링을 할 수 있다.

그러나, 사실 이러한 원칙은 주로 스케쥴링이나 자원 할당을 담당하는 운영체제에서 필요로 하는 면을 지원하는 성격이 강하다. CPU나 기억장치 또는 버스의 대역폭은 하드웨어적으로 고정되는 것이고, 그들을 어떻게 운용하는가에 따라 실제 수행시간이 결정되기 때문이다.

그러면 기억 장치와 I/O의 예를 들어 분할과 예측가능성에 대해 살펴보겠다.

(1) 기억 장치 분할

작은 크기의 기억 장치로도 많은 작업을 수행하고 성능을 향상시키기 위해 대부분의 컴퓨터 시스템에서 사용하고 있는 가상 기억 장치, 혹은 계층적 기억 장치는 기억 장치에서 자료를 가져오는 시간이 고정되어 있지 못하기 때문에 실시간 시스템에는 거의 사용하지 않았다. 캐쉬의 경우 작업 집합(working set)이 바뀔때는 적중률(hit ratio)이 평소와 달라 수행시간이 길어지게 되는데 그것은 시스템내 다른 작업과의 관계 등 부하의 동적 특성에 따라 매번 다르기 때문에 수행시간을 캐쉬가 없는 것처럼 계산해야만 한다.

이 경우 분할의 원칙을 적용시켜보면 기억장소를 여러개의 (여러 크기의) 고정된 구획으로 나누고 OS 커널 등 자주 수행되는 작업에 대해 전용 구획을 지정해서 사전 할당(preallocating)을 하는 방법을 생각해 볼 수 있다. 이는 유연성을 감소시켜 자원의 낭비를 초래할지라도 기억 장소 접근 시간이 항상 예측가능하기 때문에 시간의 절약을 기대할 수 있다. 이러한 방법의 예로 SMART 캐쉬[6,7]를 들 수 있다.

(2) I/O 분할

입출력과 관계된 주요점은 DMA의 사용, 감지기, 인터럽트 처리 등이다. DMA는 CPU의 사이클을 잠시 입출력에 사용하는 사이클 스틸링(cycle stealing)과, 각 기억장치 사이클을 CPU를 위한 타임 슬롯과 DMA를 위한 타임 슬롯으로 미리 할당하여 사용하는 기억 장치 시간 분할(time slice) 방법이 있다. 물론 시간 분할 방법이 응답시간을 정확하게 예측할 수 있는 방법이며, 기억 장치 사이클에 대해 분할의 원칙이 적용된 것이라고 할 수 있다.

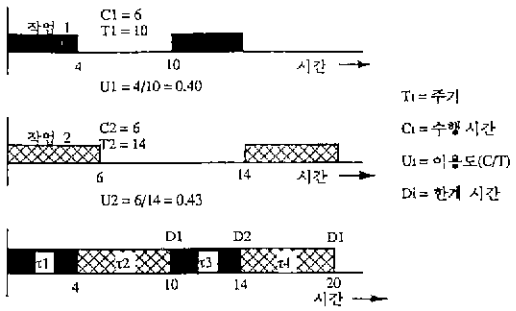
감지기(sensor)는 입력 장치로서 대부분의 실시간 시스템들이 많은 수의 감지기를 포함하고 그를 외부 상황과 자료를 계속 입력하는데 사용한다. 그리고, 입력된 정보는 시스템의 목적에 맞도록 처리되어야

한다. 그러나, 감지기의 수가 많다는 점과 입출력에 걸리는 시간이 크다는 점이 문제가 되므로, 분할의 원칙을 여기에 적용한 결과는 정보를 입력하는 전용 처리기를 정보를 처리하는 주처리기와 분리하는 것이다. 그리고, 주처리기의 사이클이나 버스의 일부분을 고정 할당시켜 입력 전용 처리기가 사용하도록 한다. 이 방식은 처리기가 더 필요하다는 비용의 문제가 있지만 전체 문제의 크기와 복잡도를 감소시켜 성능을 향상시키는 효과를 가져온다. MAFT[13]에서 OS 커널을 수행하는 처리기와 응용 프로그램을 수행하는 처리기를 분리한 것도 비슷한 예가 된다.

인터럽트는 어떠한 시스템에서도 발생 가능하며 현재 수행되는 태스크에 예측 불가능한 지연을 일으키는 원인이 된다. 특히, 실시간 시스템의 대부분이 인터럽트 위주로 운용되는 경우가 많으므로 이를 잘 처리해야만 한다. 그러나, 일반적인 시스템에서와 같이 인터럽트 처리를 가장 우선순위가 높은 태스크로 처리해서는 안되는데, 이는 다른 응용 프로그램 혹은 태스크의 시간 한계를 어길 가능성이 있기 때문이다. 이 경우 사용할 수 있는 방법은 응용 태스크, 인터럽트 태스크 등의 각각에 대해서 서로 다른 스케쥴러를 사용해 인터럽트 처리 자체를 스케쥴링하는 것이다. 이러한 스케쥴러들은 하나의 알고리즘으로 구현될 수도 있고 그렇지 않을 수도 있지만 모든 태스크들의 한계 시간을 지킬 수 있도록 조화를 이루어야 한다.

또 한가지, 인터럽트와 그 처리 시간을 예측 가능하게 만드는 방법은 인터럽트를 하나의 특발성(sporadic) 태스크로 간주하여 인터럽트 처리 루틴을 항상 활성화된 상태로 두는 방법이 있다. 특발성 태스크란 도착 시간 사이의 간격이 가장 짧은 주기적 태스크를 말한다. 즉, CPU 시간의 일부분을 언제나 인터럽트에 할당해 두는 것이다. 사실 설계시에 이미 어떠한 인터럽트가 발생할 가능성이 있으며 그의 처리는 어떻게 해야 한다는 등이 결정되므로, 이러한 스케쥴링이나 CPU 시간의 할당 방식은 그에 따르는 손실은 있지만 가능한 방법이다.

이상의 예에서도 보다시피 예측가능성을 보장하기 위해서는 될 수 있는대로 정적이고 고정된 형태의 구조가 되기 쉬운데 현재 응용들에서 요구하고 있는 분산된 동적 구조와 어떻게 적절하게 배합시켜 원하는 성능을 얻을 것인가가 문제이다. 또한 자원의 낭비와 예측가능성의 보장 간의 상충점도 고려되어 적절한 효율과 실시간성을 얻을 수 있어야 한다.



(그림 1) 0.83의 이용도를 가지는 두 작업의 스케쥴링

2.2.2 고성능

고성능 처리기 및 기억장치를 사용하여 태스크의 수행 시간을 단축시킬 수 있다는 것은, 보다 많은 태스크를 같은 시간 내에 수행할 수 있음으로써 스케쥴 가능성(schedulability)을 증가시켜 실시간 응용의 한계 시간을 보다 잘 지킬 수 있게 됨을 의미한다. 예를 들어, 작업 1이 4의 계산 시간을 가지고, 작업 2는 6의 계산 시간을 가지며, 그 주기는 각각 10, 14인 경우의 스케쥴링과 이용도(utilization)를 (그림 1)과 같이 생각할 수 있다.

이 경우 둘보다 많은 수의 태스크가 있다면 스케쥴링이 불가능하게 된다. 여기에서 태스크의 수행시간을 줄임으로써 이용도를 감소시킬 수 있다면 그만큼 다른 태스크를 스케쥴할 수 있다. 물론, 해야 할 작업의 수가 적다면 굳이 그러한 것을 고려할 필요가 없으나, 대부분의 경우 많은 작업이 존재하고 수시로 발생하는 돌발 상황이 있기 때문에 성능을 향상시키는 것이 필요하다. 특히, 실시간 시스템에서는 예측 가능성을 위해 많은 자원들의 낭비가 발생하므로, 보다 효율적인 자원 사용의 방향으로 성능을 향상시키는 방안을 연구하는 것이 필요하다.

2.3 적응성

최근에 있어서의 기술 발전의 추세와 실시간 시스템이 적용될 수 있는 응용 분야의 확대 현황을 고려할 때, 실시간 시스템이 그러한 변화에 대한 적응성을 갖추어야 한다는 것은 점차 필수적인 요구 사항이 되고 있다. 아키텍처에 있어서의 적응성은 각 구성 요소의 적응성(component adaptability)과 시스템의 적응성(system adaptability)으로 나누어 생각해 볼 수

있다.

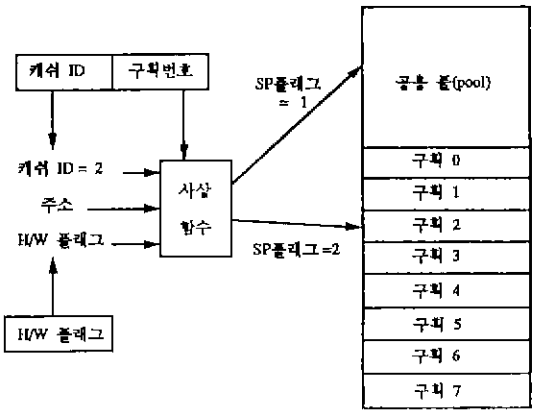
구성요소 적응성은 시스템을 구성하는 CPU, 기억장치, 버스 제어기 등 각 요소가 범용성 또는 적응성을 가지도록 설계되어 각각의 응용구조에 적당한 연결구조(topology)를 갖는 시스템의 구성이 가능하도록 하는 것이다. 이는 시스템의 각 구성요소들이 빠른 속도로 발전하고 있기 때문에 오랜 기간을 두고 사용하는 경우가 많은 실시간 시스템의 경우, 이러한 발전된 장치를 그때 그때 수용하여 이용할 수 있어야 하기 때문이다.

시스템의 적응성은 다시 두 가지로 나눌 수 있는데, 그 하나는 환경에 대한 적응성, 또 하나는 응용에 대한 적응성으로 생각할 수 있다. 같은 응용 분야라 할지라도 사용되는 환경이 변화할 수 있으므로 변화하는 환경에 적응할 수 있는 능력이 있어야 한다. 또 실시간 시스템이 대부분 특수 목적으로 구성되기는 하나 비슷한 분야의 서로 다른 응용 시스템으로 이용될 수도 있으므로, 미리 그에 대한 것들을 인자화하는 등의 수법을 사용하여 큰 수정없이 적응할 수 있도록 하는 것이 필요하다. 이는 개발 비용이나 노력을 절감시킨다는 측면에서도 필요한 기능이다.

구성요소 적응성은 하드웨어 수준에서 적응성을 제공하는 경우가 많으며 시스템 적응성은 소프트웨어로 구현되는 경우가 많다. 물론 시스템 적응성을 지원하기 위해서는 하드웨어의 변화를 미리 내장하고 기본 명령으로 갖추고 있어야 하는 경우도 있다. 간단한 예로 SMART 캐쉬의 set/reset 명령을 생각할 수 있다.

구성요소의 적응성에 대한 가장 극단적인 경우는 기존의 상업화된 마이크로프로세서, 상업화된 OS, 표준 인터페이스, 표준 버스 등을 사용하여 아키텍쳐를 구성하고 소프트웨어로 실시간성을 제공하는 것이다. 이 방법은 빠른 개발 시간과 가장 유연한 적응성을 보장할 수 있으나 특수 목적에 대한 효율적이고도 높은 성능을 얻기가 어렵다는 단점이 있다. 그래서, 실시간 시스템이나 고장허용 시스템의 연구에서 구성요소 적응성을 고려할 때에는 대개 몇몇 응용 분야에 대해 적응성을 가지는 기본 블럭을 설계하고 그를 시스템으로 구성하는 접근 방식을 택한다. 이러한 접근 방식의 예로써 FTBBC[11], MMFCS[14,1,2] 등이 있다.

하드웨어 측면에서의 시스템 적응성은 아직 운영체제 커널이나 그 외 소프트웨어에서 요구하는 바를 뒷받침해 주는 정도라고 할 수 있다. 응용에 대한



(그림 2) SMART 캐쉬 분할을 위한 하드웨어 구조

시스템 적응성은 구성요소를 블럭화하는 방법으로 역시 구현될 수 있다.

III. 사례 시스템 연구

3장에서는 2장에서 연구된 실시간 시스템의 요건 중, 특히 예측가능성과 적응성을 고려하여 설계된 시스템 사례들에 대해 살펴보았다. 먼저 SMART 캐쉬는 기억 장치에 분할의 원칙을 적용시켜 예측가능성을 높이고자 한 예이고, 모듈 접근 방식은 구성요소 적응성을 중점적으로 고려하여 설계된 시스템으로 응용에 대한 시스템 적응성도 만족하는 예이다.

3.1 SMART

SMART(Strategic Memory Allocation for Real-Time) 캐쉬[6,7]는 우선 순위 기반의 preemtive 스케쥴링을 사용하는 실시간 시스템에서, 캐쉬를 분할하여 특정 태스크에 각 구획을 할당함으로써 계층적 기억장치 구조가 갖는 비예측성을 없애고, 캐쉬를 이용한 성능 향상을 꾀한 것이다. SMART에서 취한 분할 방식은 (그림 2)와 같다.

여기서 플래그는 실시간 캐싱과 일반적 캐쉬 동작을 사용자가 지정할 수 있도록 하는 역할을 한다. 캐쉬 자체는 직접 사상 캐쉬(direct mapped cache)[4]나 연관 캐쉬(set associative cache) 중 아무것이나 가능하다. 주소는 캐쉬 ID 플래그를 결합하여 얻어지는데 캐쉬 ID는 문맥 교환(context switch)시에 ID 레지스터로 들어온다. 플래그는 기계 명령어 set/reset으로 조절 가능하다.

자주 수행되는 작업이나 중요도가 큰 작업은 전용 구획을 하나씩 할당받는다. 그리고 수행되고 있지 않은 경우에도 그 구획을 다른 작업에 빼앗기지 않아 데이타를 그대로 유지할 수 있어 언제나 동일한 적중률을 보장할 수 있다. 공용 풀(pool)은 중요하지 않은 작업들에 대해 일반 캐쉬와 같은 작용을 한다. 여기서는 예측가능성이 없어지나 작업들이 엄격한 시간엄수를 요구하지 않으므로 문제가 되지 않는다. 전용구획은 하나의 작업이 아니라 같은 우선 순위를 가지는 태스크들에 대해 할당될 수도 있다. 이때는 우선순위가 같으므로 preemption이 일어나는데 이로 인해 예측가능성에 손상이 간다.

SMART 캐쉬를 실제 MIPS R3000 프로세서에 적용시켜 설계하고 캐쉬 할당을 위한 VDA(Vaule Density Acceleration) 알고리즘을 개발하여 성능을 측정한 연구가 또한 발표되었다[7]. 한 태스크에 여러 개의 캐쉬 분할 구역을 할당할 경우 필요한 캐쉬 구역을 선택할 수 있도록 하는 하드웨어를 추가하는 부담과 실행시에 선택에 따른 시간 지연이 일어나므로, 이를 간단하게 만들기 위해 한 태스크에 할당할 수 있는 캐쉬 분할 구역의 수를 2의 승수로 제한하여 단순한 멀티플렉서로 구현하였다. 이러한 구현으로 적절한 캐쉬 사용빈도와 적중률을 가정할 때, 캐쉬가 없을 때보다 약 50%의 성능 향상을 보일 수 있음이 분석되었고, 멀티플렉서로 인한 지연은 약 5%인 것으로 나타났다. VDA 알고리즘은 작업의 수행 빈도, 기억장치 접근 회수, 구역 크기에 따른 적중률의 변화 등의 인자를 측정하여 점수를 매기고 그 점수에 따라 작업에 할당하는 캐쉬의 크기를 조절하는 알고리즘이다. Polynomial 시간내에 수행가능하고, 최적 결과와 비교해 약 1% 정도 어긋나는 성능을 보인다.

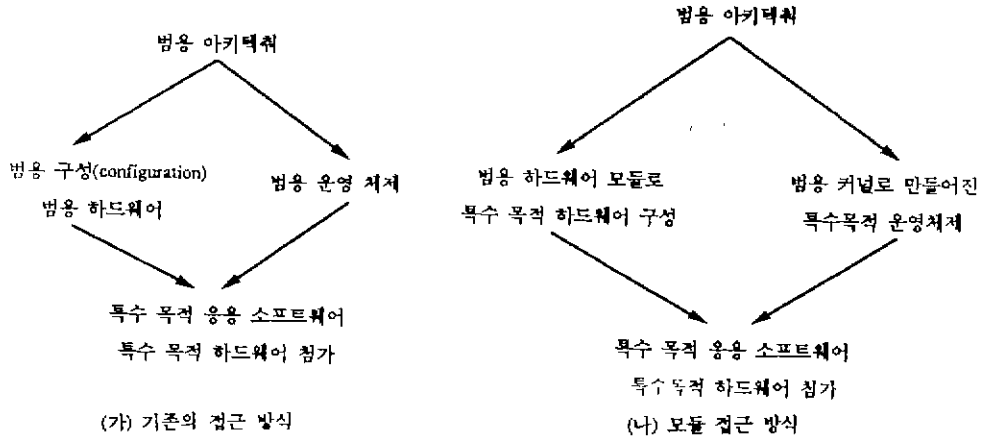
3.2 적응성을 고려한 모듈 접근 방식

Arnold[1] 등에 의해 제안된 시스템은 (그림 3)과 같은 접근 방식을 기본으로 하고 있다.

(그림 3)의 (가)는 문제를 하드웨어에 맞추는 방식인데 비해 실시간 시스템은 대개 그 응용이 고정되어 있는 경우이므로 범용 모듈로써 응용에 알맞는 특수목적 아키텍쳐를 구성하는 것이 합리적이다. 이 연구에서는 고속, 고장허용, 실시간 처리 특성을 내장한 기본 하드웨어 모듈들을 정의하고, 분산 혹은 다중처리 시스템의 여러 가지 형태를 구성할 수 있음을 보였다.

3.2.1 하드웨어 모듈

- CPU: 16비트 범용, 1M 주기억 장치, 1MIPS의 성능을 가지며 명령이 추가될 수 있다.
- MDS(Multiple Data Stream Adjunct): CPU에 부착되는 2개의 slave ALU로 구성되어 있으며, CPU에 FORK, JOIN 명령을 추가함으로써 SIMD(Single Instruction Multiple Data) 모드로 구성될 수 있게 한다.
- VOA(Vector Operations Adjunct): 15 MIPS의 고속 파이프라인 수치연산기로 구성된 배열 처리기(array processor)이며 MDS와 마찬가지로 CPU에 적절한 명령어를 추가하여 사용할 수 있다.
- LM(Local Memory): 16비트의 4K, 16K 혹은 32



(그림 3) 실시간 시스템 구성 방식

K 워드로 된 3-port 기억장치이며 250 ns의 사이클 타임을 가진다.

• DBM(Common Bulk Memory): 전역 버스를 통해 접근 가능한 공용 기억장치이며, LM보다 접근시간이 느리다. 보통 여러개를 묶은 books로 구성되고 각 books은 BIU에 의해 제어된다.

• BIU(Bus Interface Unit): 한 노드당 3개까지의 전역버스(GB)를 연결하는 기능을 가진다. 버스는 서로 다른 노드에서 수행되는 태스크간 또는 태스크와 CBM 간에 독립적인 통신을 제공하며, 버스 할당은 결정적(deterministic)이며 분산된 토큰 전달 방식을 사용하여 각 노드에 대한 버스 대역폭을 보장해 준다.

• IOC(Input/Output Controller): BIU와 노드의 지역 버스들을 연결하여 노드와 다른 노드, 혹은 CBM간의 인터페이스를 제공한다. IOC와 BIU는 시스템의 연결 요소로서 시스템 상에서 수행되는 프로세스들에 대해 일관된 통신 방법을 제공한다.

3.2.2. 시스템 구성례

앞절에서 설명한 각 모듈들을 사용하여 원하는 연결망 구조를 가지는 시스템을 구성할 수 있다. CPU의 명령이 확장 가능하고 시스템 소프트웨어로써 여러가지 아키텍쳐를 지원할 수 있으며, 각 버스의 연결이 특수한 제어기로써 통제되어 여러가지 시스템 구조를 가질 수 있다. 그 중, 일반적인 다중 처리기가 어떻게 구성되는가를 (그림 4)의 예에서 볼 수 있다.

특수한 응용분야에 대한 시스템을 구성할 때는 먼저 그 응용에 대해 정확한 요구 분석과 성능분석을 한 후, 적절한 아키텍쳐를 구성할 수 있다. 예를 들어, 미사일 또는 우주 방위 시스템을 생각해 보면, 이는 많은 양의 입력 데이타에 대해 filtering과 신호 처리(signal processing)를 행하는 것이므로, 감지기(sensor)의 동기성 여부에 따라 SIMD 또는 MIMD인가를 결정하고, 신호 처리를 위한 벡터 처리 능력을 가지는 아키텍쳐로 구성해야 한다.

IV. 설계시의 주요 논점

실제로 처리기 아키텍쳐를 설계하는 데에는 일반적인 처리기 설계와 마찬가지로 여러 가지 결정해야 할 문제와 상충점들이 있다. 대표적인 몇 가지 문제를 들면 다음과 같다.

• 특수목적(special-purpose) 아키텍쳐, 혹은 범용(general-purpose) 아키텍쳐

• 단일 처리기로 구성할 것인가? 아니면 다중 처리기로 구성할 것인가?

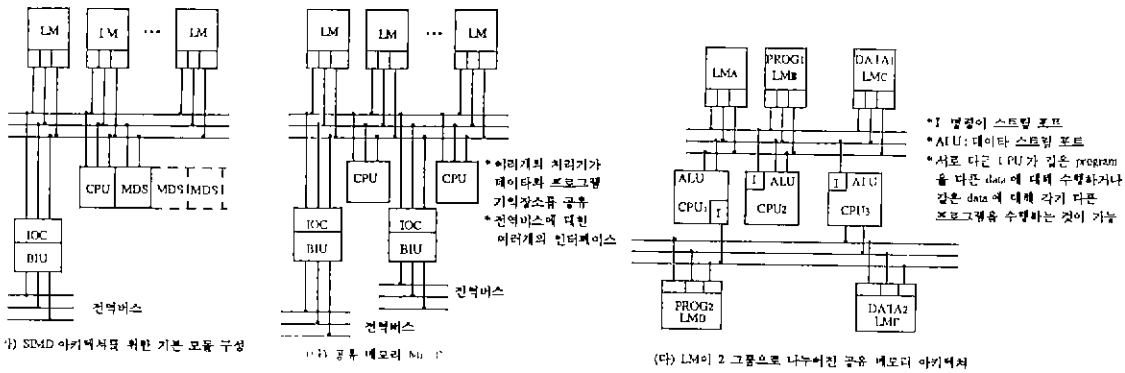
• 기억 장치 계층 구조 및 입출력은 어떻게 할 것인가?

• 다중 처리기의 경우, 각 노드는 단일 처리기인가?, 다중 처리기인가?

• 다중 처리기라면 노드 수는 얼마나 할 것이며, 수행 모델은 어떤 것으로 할 것인가?

그 외에도 다중 처리기를 설계할 시에는 각 노드의 구조, 연결망의 구조(interconnection topology) 등 결정해야 할 많은 문제들이 있다.

이러한 문제들은 대개 그 시스템이 적용될 응용분야와 그에 따른 세 요건의 요구 수준에 의해 해답이 결정된다. 지금까지의 실례를 보면, 실시간 처리기가 다른 시스템의 한 구성요소인 경우가 많기 때문에 대부분의 실시간 시스템이 범용이라기보다는 특수목적이었다. 또한, 일반적인 컴퓨터 시스템에서 성능을 향상시키기 위해 도입하는 계층적 기억 장치 구조,



(다) LM이 2 그룹으로 나누어진 공유 메모리 아키텍쳐

(그림 4) 모듈 접근 방식을 사용한 시스템 구성의 예

I/O를 위한 DMA 방식, 프로세스 동적 할당 등은 그 특성상 수행 동작 양태를 정확히 예측하기가 힘들기 때문에 실시간 시스템에서는 거의 사용되지 않는다.

그러나, 현재는 VLSI 기술이 발달하고, 응용 분야도 많은 계산량과 보다 엄격한 관리체재를 요구하고 있기 때문에 기본적인 문제에 대해서는 그 추세가 결정되는 상태이다. 즉, 고성능 처리기를 구현하기 위해 다중 처리가 일반화되고 있으며, 제어해야 할 시스템이 분산되어 있는 관계로 분산 처리를 위한 연구도 많이 진행되고 있다. 또한, 전체 구조는 특수목적에 적합하게 조정할 수 있으나, 노드 처리기, 통신용 인터페이스 등은 범용으로 구성하여 유연성을 제공하는 시스템도 있다. 그래서, 현재 실시간 아키텍쳐에서의 주요 논점은 다음과 같다[12].

- 실시간 처리용 다중 처리기를 구성하는 데 있어서의 연결망 구조
- 프로세스간의 빠르고 예측 가능한 통신 방법
- 운영 체제에 대한 하드웨어 지원
- 고장 허용 특성

그리고, 앞에서 나열한 설계시의 문제들을 다시 정리하면 다음과 같다.

- 단위처리기의 구조

실시간 처리를 위한 단위 처리기는 특수 용도로 설계될 수도 있고 아닐 수도 있다. 또, 시간의 측정이나 스케쥴링, 동기화 등 시간 한계를 지키기 위해 시스템 소프트웨어나 언어에서 요구하는 특징을 하드웨어 수준에서 구현하는 것이 필요한 경우도 있다. 그리고, 원하는 작업을 제 시간에 수행할 수 있도록 응용 분야별로 적당한 속도를 낼 수 있어야 한다. 또한, 대부분 감지기와 실행기 사이의 많은 입출력을 필요로 하기 때문에 고성능의 I/O 인터페이스가 있어야 한다. 또, 신뢰성을 향상시키기 위한 중복성(redundancy)의 도입이 필요한 경우도 있다.

- 기억 장치 계층 구조

대부분의 실시간 시스템이 예측가능성을 보장하기 위해 계층적 기억 장치를 사용하지 않고 있지만, 성능 향상을 위해 캐쉬 등을 도입해야 할 필요가 있다. 예측 가능성을 손상시키지 않으면서 보다 높은 성능을 얻을 수 있는 방법이 있어야 한다.

- 처리기와 기억 장치, 입출력 부분에 대한 연결망 구조

성능의 향상, 응용분야에서의 적합성들을 고려하는 적절한 연결망 구조가 필요하다. 특히, 실시간 응용에서는 많은 양의 데이타의 입출력이 필수적이다. 현재까지 처리기와 기억 장치간의 연결망 구조는 많은 연구가 되었지만, 효율적인 입출력을 위한 연결망의 구조에 대한 연구는 거의 없는 실정이다.

- 고속, 고신뢰도의 통신기법

서로 다른 노드에 있는 프로세스나 태스크들이 고도의 신뢰성을 가지고 고속 통신을 할 수 있어야 하며, 그의 특성이 완전히 알려져 있어 성능을 예측할 수 있어야 한다. 예를 들어, virtual cut-through[3]와 같은 방식을 생각해 보면, 적절한 작업량이 부과되었을 경우는 고속 통신을 보장할 수 있으나, 과부하의 경우 통신 지연이 일어나고 그것을 정확하게 예측할 수 없기 때문에 실시간 시스템에 적용하기에는 어려운 면이 있다.

- 오류 처리를 위한 아키텍쳐의 지원

신뢰성을 향상시키기 위해 오류(fault)의 발견, 재형성(reconfiguration), 빠른 회복(recovery) 등을 제공할 수 있어야 한다.

- 스케쥴링 기법과 운영체제에 대한 하드웨어의 지원

스케쥴링에서 요구하는 빠른 preemptability, 우선순위 관리에 필요한 우선 순위 큐, 우선 순위의 변경 및 지정 등을 하드웨어로 지원할 수 있어야 한다. 그리고, 운영 체제에서 구현되는 실시간 프로토콜, 다중 문맥(contexts), 실시간 기억 장치 관리 등에 대해서도 하드웨어로 지원이 필요한 요소가 있다.

- 실시간 언어에 대한 아키텍쳐 지원

각 작업의 시간 관계를 표현하거나, 스케쥴링에 필요한 작업 시간의 계산을 수행하고 필요한 정보를 추출하는 컴파일러 등이 가능하기 위해 하드웨어 아키텍쳐로 그를 지원할 수 있어야 한다.

V. 결 론

본 연구에서는 실시간 시스템을 구성하는 아키텍쳐의 요건과 설계시의 주요 논점에 대해 기술하고, 각 요건을 고려한 실시간 시스템의 사례들을 살펴보았다.

실시간 시스템이 갖추어야 할 요건으로 정확성, 시간 엄수성, 적응성 등에 대해서 살펴본 결과, VLSI 및 하드웨어 기술이 발전함에 따라 고속, 고성능의 처리기를 개발하거나 고장허용성을 지원하는 것은 더욱 용이해지고 있으나, 실시간 시스템에서 절대적으로 요구되는 시간 엄수를 위한 예측가능성의 구현은 아직 미비한 상태임을 볼 수 있었다. 또한, 예측가능

성을 고려하는 것은 성능 향상을 위한 방법을 도입하는 것과 상충되는 면이 있으며, 시스템 자원 이용이 비효율적으로 되는 경향을 나타낸다. 적응성은 각종 응용이 복잡해지고 다양해지는 추세와 하드웨어 기술의 발전에 따라 앞으로 더욱 많이 고려되어야 할 측면이라고 하겠다.

실제 실시간 시스템의 구성은 시스템이 적용되어야 할 응용분야의 요구 사항에 가장 크게 좌우된다. 어느 정도의 시간엄수 요구를 가지는가? 어느 정도의 신뢰성을 필요로 하는가? 어느 정도의 적응성이 필요한가? 비용을 어느 정도 투자할 수 있는가? 등이 응용 분야에 대해 분석되어야 한다. 그리고, 이들간의 상충점을 공학적 접근 방식에 의해 해결해야 한다. 실시간 시스템 구성시 아키텍쳐를 최적화시키는 방법에 대해 연구한 [9,10]와 같은 논문들도 있다.

참 고 문 헌

1. R. G. Arnold, R. O. Berg, and J. W. Thomas. A Modular Approach to Real-Time Supersystems. *IEEE Transactions on Computers,* **31**(5), pp. 385~398, May 1982.
2. C. Constantinescu and C. Sandovici. A Fault-Tolerant Microcomputer for Advanced Control: Architecture and Performability Analysis. *In Proceedings of the Real-Time Systems Symposium,* pp. 222~228, IEEE Computer Society Press, 1989.
3. W. J. Dally and C. L. Seitz. The Torus Routing Chip. *Distriuted Computing,* **1**(3), 1986.
4. M. Hill. A Case for Direct Mapped Caches. *IEEE Computer,* pp. 25~40, Dec. 1988.
5. B. W. Johnson. *Design and Analysis of Fault-Tolerant Digital Systems.* Addison-Wesley Publishing Company, 1989.
6. D. B. Kirk. SMART(Strategic Memory Allocation for Real-Time) Cache Design. In *Proceedings of thd Real-Time Systems Symposium,* pp. 229~237. IEEE Computer Society Press, 1989.
7. D. B. Kirk and J. K. Strosnider. SMART(Strategic Memory Allocation for Real-Time) Cache Design Using the MIPS R3000. In *Proceedings of the Real-Time Systems Symposium,* pp. 322~330. IEEE Computer Society Press, 1990.
8. H. D. Kirrmann and F. Kaufmann. Poolpo-A Pool of Processors for Process Control Applications. *IEEE Transactions on Computers,* **33**(10), pp. 369~378, Oct. 1984.
9. C. M. Krishna, K. G. Shin, and I. S. Bhandari. Processor Tradeoffs in Distributed Real-Time Systems. *IEEE Transactions on Computers,* **C36** (9), pp. 1030~1040, Sep. 1987.
10. A. Pedar and V. V. S. Sarma. Architecture Optimization of Aerospace Computing Systems. *IEEE Transactions on Computers,* **32**(10), pp. 911~922, Oct. 1983.
11. D. A. Rennels, A. Avizienis, and M. Ercegovac. A Study of Standard Building Blocks for the Design of Fault-Tolerant Distributed Computer Systems. In *Proceedings of the 1978 International Symposium on Fault-Tolerant Computing,* pp. 144~149, Jun. 1978.
12. J. Z. Stankovic and K. Ramamritham. *Tutorial Hard Real-Time Systems.* IEEE Computer Society Press, 1988.
13. C. J. Walter, Kieckhafer, R.M. and A. M. Finn. MAFT: A Multicomputer Architecture for Fault-Tolerance in Real-Time Control Systems. In *Proceedings of the Real-Time Systems Symposium,* pp. 133~140, 1985.
14. J. A. White, G. L. Hartmann, R. E. Pope, and J. W. Hunger. *A Multi-Microprocessor Flight Control System.* Honeywell Systems and Research Center, contract No. AFWAL-TR-81-3044, May 1981.

윤 현 수

1979 서울대학교 공과대학 전자공학과(학사)
1981 한국과학기술원 전산학과 석사학위 취득
1981 ~1984 삼성전자 연구원
1988 오하이오 주립대학 전산학 박사학위 취득
1988 ~1989 AT & T Bell Labs. 연구원
1989 ~현재 한국과학기술원 전산학과 부교수
관심 분야 : 병렬 컴퓨터 구조, 뉴럴 네트워크, 실시간 시스템.

윤 현 주

1988 서울대학교 컴퓨터공학과(공학사)
1990 한국과학기술원 전산학과 석사학위 취득
1990 ~현재 한국과학기술원 전산학과 박사과정
관심 분야 : 병렬 컴퓨터 구조, 그래픽 하드웨어, 멀티미디어, 실시간 시스템