論文96-33B-6-3

NUMA 다중처리기에서 조정가능한 지연 카운터를 이용한 페이지 복사 기법

(Page Replication Mechanism using Adjustable DELAY Counter in NUMA Multiprocessors)

李 鍾 雨 *, 曺 裕 根 *

(JongWoo Lee and Yookun Cho)

요 약

NUMA(Non-Uniform Memory Access) 구조의 공유 메모리 다중처리기 시스템에서 참조 국지성의 활용은 병렬 처리의 성능에 큰 영향을 미친다. 본 논문에서는 운영체제가 참조 국지성을 관리하는데 도움을 주기 위한 개선된 하드웨어 메모리 참조 카운터를 제시한다. 제시된 참조 카운터 방식에서는 기존의 참조 카운터들과는 달리 운영체제의 페이지 복사 정책을 다양한 메모리 참조 패턴에 적응시키기 위해 카운터의 값이 동적으로 그리고 주기적으로 조정된다. 우리는 실제 병렬 응용 프로그램들을 사용한 실행-구동형 시뮬레이션을 통해 제시된 ''조정가능한 지연 카운터''가 이들의 성능에 미치는 영향을 평가하였다. 이 성능평가를 통해 ''조정가능한 지연 카운터''를 이용한 메모리 복사 정책이 기존의 카운터를 이용한 정책보다 나은 성능을 보인다는 것과 시뮬레이션에 사용된 대부분의 병렬 응용 프로그램에 대해 고른 성능을 나타낸다는 것을 확인하였다.

Abstract

The exploitation of locality of reference in shared memory NUMA multiprocessors is one of the important problems in parallel processing today. In this paper, we propose a revised hardware reference counter to help operating system to manage locality. In contrast to the previous one, the value of counter can be adjusted dynamically and periodically to adapt the page replication policy to the various memory reference patterns of processors. We use execution-driven simulation of real applications to evaluate the effectiveness of our adjustable *DELAY* counter. Our main conclusion is that by using the adjustable *DELAY* counter the *t normalized average memory access costs* and the variance of them become smaller for most applications than the previous one and more robust memory management policies can be provided for the operating systems.

I. 서 론

NUMA 구조를 갖는 다중처리기 시스템의 운영체제에서 참조 국지성(locality of reference)을 이용하는 기본적인 메커니즘은 처리기가 자주 이용하는 페이지들을 그 처리기의 로컬 메모리에 복사하거나 이주시키는 것이다. 그러나 페이지들을 복사하는 경우 같은 페이지의 여러 복사본이 동시에 존재하게 되어 메모리 일관성(coherence) 문제가 발생한다. 즉, 어떤 처리기가 어떤 페이지의 로컬 복사본(local copy)을 수정했을 경우 그 페이지의 복사본을 보유하고 있는 모든 처리기에게 그 사실을 알려 메모리 일관성을 유지해야 한다. 메모리 일관성을 유지하는 방식에는 무효화(write-invalidate) 방식과 갱신(write-update) 방식이 있는데, 무효화에 기반한 방식에서는 변경된 페이지와 같은 시스템 내의 모든 복사본들이 무효화(inva-

* 正會員, 서울大學校 컴퓨터工學科

(Department of Computer Engineering, Seoul National University)

接受日字:1995年4月18日, 수정완료일:1996年5月13日

lidation)되고, 이후에 처리기들이 무효화된 페이지를 참조하면 페이지 폴트(page fault)가 발생하기 때문에 이때에 새로운 최신 복사본을 요구할 수 있다. 갱신에 기반한 방식에서는 변경된 페이지와 같은 복사본을 가지고 있는 모든 처리기들에게 메시지가 전달된다. 이 메시지에는 갱신된 데이타 값과 메모리 내에서의 위치가 포함되어 있어서 메시지를 받은 처리기들은 자신의 복사본을 갱신할 수 있다. 일반적으로 무효화 방식이 갱신 방식 보다 간단하기 때문에 다중처리기 시스템들에서 자주 이용되어 왔으나 만족할만한 성능을 보이지 못하고 있는데, 그 이유는 페이지 전체를 복사하는데 드는 비용과 페이지 무효화에 드는 비용, 그리고 거짓 공유(false sharing)[11]으로 인한 비용이 매우 크기 때문이다. 반면에 갱신 방식은 몇몇 상황에서 좀 더 나은 성능을 보이고 있다[2].

NUMA 구조의 다중처리기 운영체제가 해야할 중요한 결정은 크게 세가지로 나눌 수 있다. 하나는 ''원격 노드에 있는 페이지의 복사 여부'' 이고, 다른 하나는 ''원격 노드에 있는 페이지의 복사 시기'' 이며, 나머지 하나는 ''복사했던 페이지(로컬 복사본)가 해당 노드에서 자주 참조되지 않을 경우 언제 없앨(unreplicate) 것이냐'' 이다. 어떤 페이지가 그 페이지를 참조하는 처리기 p의 로컬 메모리로 복사된 이후에는 처리기 p는 그 페이지를 로컬하게 참조할 수 있기 때문에 성능 향상을 기대할 수 있다. 그러나 페이지를 복사하는데는 페이지 전송에 드는 비용과 운영체제 오버헤드가 존재하고, 또 복사한 이후에는 다른 처리기들에 의한 갱신 비용이 발생하기 때문에(갱신에 기반한 일관성 유지 방식을 이용할 경우) Markatos 등은 ''페이지는 복사의 잇점이 그 경비를 상쇄시킬 수 있을 때에만 복사되어야 한다''고 주장하고 있다[3]. 그러나 페이지 복사로 인해 성능 이득을 얻을 것인지 아닌지를 운영체제가 미리 안다는 것은 불가능하므로 기존의 연구들에서는 일정한 초기값을 주고 그 값이 0이 될 때까지 1씩 감소하는 하드웨어 메모리 참조 카운터에 기반한 페이지 복사 정책들을 제시하였다.

본 논문에서는 기존 하드웨어 메모리 참조 카운터의 기능과 운용 방식을 개선하고, 개선된 하드웨어 참조 카운터를 이용한 페이지 복사 정책을 제시하고자 한다. 한편 본 논문은 하드웨어 캐쉬가 지원되지 않는 순수 NUMA 구조의 버스-기반 공유 메모리 다중처리기 시스템을 실험 환경으로 가정하였으며 따라서 메모리 일관성의 단위도 가상 메모리(virtual memory) 시스템에 기반한 가상 페이지로 설정하였다.

Ⅱ. 기존 관련 연구

Bolosky 등[1]은 지연(*DELAY*) 카운터를 제시하였다. 이 카운터는 원격(remote) 페이지[1]마다 존재하는 하드웨어 카운터로써 어떤 처리기가 원격 노드에 있는 어떤 페이지를 최초로 참조할 때 일정한 값으로 지정된다. 이후에 그 처리기가 그 페이지를 원격으로 참조할 때마다 그 페이지의 지연 카운터 값은 1씩 감소되고, 카운터 값이 0으로 될때에 운영체제에 인터럽트가 발생하여 운영체제는 그 페이지를 로컬 메모리로 복사한다. 이 지연 카운터 메커니즘은 최근의 메모리 참조 패턴이 가까운 미래에도 유지될 것이라는 것을 가정하고 있다. 자주(지연 카운터의 초기값의 횟수 만큼) 참조된 페이지만을 로컬 메모리로 복사하겠다는 것이다. 또한 지연 카운터에 의해 잦은 페이지 복사를 방지하여 시스템의 성능을 향상시키고자 하였다.

한편 Markatos와 Chronaki[3]는 갱신(*UPDATE*) 카운터를 제시하였는데 갱신 카운터는 로컬 메모리로 복사해온 로컬 복사본의 일관성 유지를 위해 발생하는 갱신 요청(다른 처리기들에 의한)의 횟수를 세는 페이지별 카운터이다. 만약 처리기 p의 로컬 메모리에 복사된 페이지를 처리기 p는 그리 자주 참조하지 않고 오히려 다른 처리기들에 의해 갱신이 더 자주 발생한다면 로컬 복사본을 유지함으로써 얻을 수 있는 이득이 상쇄되어 시스템 성능에 나쁜 영향을 끼칠 수 있다. 운영체제는 로컬 메모리에 존재하는 페이지 복사본들에 대해 이 카운터의 값을 일정한 초기값으로 설정한 후 상호 연결망(interconnection network)으로부터 갱신 요청이 들어올 때마다 해당 페이지의 갱신 카운터 값을 1씩 감소시킨다. 갱신 카운터 값이 0으로 되면 운영체제에 인터럽트를 발생시켜 운영체제로 하여금 그 로컬 복사본의 제거(unreplication) 여부를 결정하도록 한다. 운영체제는 그 로컬 페이지에 대한 로컬 참조의 횟수보다 다른 처리기들에 의한 갱신의 횟수가 더 많으면 그 페이지를 없애버린다. 그렇지 않은 경우는 갱신 카운터 값을 다시 초기값으로 지정한다. 갱신 카운

1) 자신의 로컬 메모리가 아닌 원격 메모리(다른 노드들의 로컬 메모리)에 존재하는 페이지

터는 로컬 메모리로 복사한 페이지가 타 처리기들의 갱신 요청에 의해 오히려 시스템의 성능을 저하시키는 것을 방지하기 위한 것이다.

그러나 위와 같은 메모리 참조 카운터들은 응용 프로그램들의 다양한 메모리 참조 패턴에 잘 적응하지 못한다는 단점이 있다. 즉, 카운터의 초기값이 시스템 내의 모든 페이지에 대해 일정하고 언제나 1씩 감소되기만 하므로 페이지 복사로 인해 이득을 본 경우와 손해를 본 경우를 구분하지 못한다. 예를 들어 지연 카운터에 의해 어떤 페이지를 로컬 메모리로 복사한 후 그 페이지에 대한 로컬 참조 횟수가 페이지 복사에 소요된 비용을 상쇄시키지 못할 정도의 횟수 만큼 발생했다면, 다음에 그 페이지를 다시 로컬 메모리로 복사할 경우나 또는 다른 처리기가 그 페이지를 복사하려고 할 때에는 좀 더 신중히 복사 여부를 고려하여야 하는데 Bolosky 등이 제시한 지연 카운터를 이용하면 이를 반영할 수 없다(이하 Bolosky 등이 제시한 지연 카운터를 고정형(non adjustable) 지연 카운터라 칭함). 한편 갱신 카운터 방식에서는 앞에서도 언급했듯이 어떤 로컬 복사본에 대한 상호 연결망으로부터의 갱신 요청 횟수와 로컬 참조 횟수(예를들어 N)를 비교하여 그 로컬 복사본의 제거 여부를 결정한다고 제안되어 있으나 실제로 N을 얻을 수 있는 방법에 대해서는 언급되어 있지 않다.

본 논문에서는 기존의 고정형 지연 카운터의 단점을 개선한 조정가능한(adjustable) 지연 카운터를 제시한다. 제시된 기법은 과거에 페이지 복사로 인해 이득을 본 페이지에 대해선 지연 카운터의 값을 줄여 이후에는 다른 노드들이 좀 더 일찍 그 페이지를 복사할 수 있도록 하였으며, 반대로 과거에 페이지 복사로 인해 오히려 손해를 본 페이지들에 대해서는 지연 카운터 값을 늘려 이후에는 그 페이지에 대한 다른 노드들의 복사 시기를 지연시킬 수 있도록 하였다. 우리는 페이지 복사로 인해 이득을 본 페이지는 이후에 다시 같은 노드나 다른 노드로 복사되더라도 역시 이득을 볼 것으로 가정하였으며 반대로 페이지 복사로 인해 오히려 손해를 본 페이지의 경우는 이후에 여러 노드로 다시 복사되더라도 역시 손해를 볼 것이라고 가정하였다. 이러한 가정은 응용 프로그램들의 메모리 참조 패턴에 따라 예외적인 경우도 있으나 지연 카운터 메커니즘에 내재한 "참조 국지성" 가정을 좀 더 확장한 것이라고 할 수 있다. 우리는 실제 병렬 응용 프로그램을 이용한 시뮬레이션(simulation)을 통해 사용한 응용마다 약간씩의 차이는 있으나 우리의 조정가능한 지연 카운터 기법이 기존의 고정형 지연 카운터 보다 나은 성능을 보인다는 것을 확인하였다(제 5장 참조). 또한 본 논문에서 제시된 카운터는 원격 페이지에 대한 지연 카운터 기능 이외에 로컬 복사본에 대해서는 로컬 참조 횟수(위에서 언급한 N)를 세는 용도로도 이용될 수 있도록 하였다.

Ⅲ. 조정가능한(Adjustable) 지연 카운터

조정가능한 지연 카운터 기법에서는 기존의 고정형 지연 카운터 방식과는 달리 초기화 후에 카운터의 값이 동적으로 그리고 주기적으로 변한다. 원격 노드에 있는 페이지를 최초로 참조할 때에 카운터는 어떤 임계값으로 초기화되고 그 카운터 값에 해당하는 횟수 만큼의 원격 참조 후에 로컬 메모리로 복사한다는 점은 고정형 지연 카운터 방식과 같으나 복사한 이후에 그 페이지의 카운터 값이 주기적으로 변한다는 점에서 기존의 방식과 다르다. 각 처리기에서 주기적으로 실행되는 **페이지 카운터 조정 디먼**(page counter scanner daemon) 프로세스는 자신의 로컬 메모리에 존재하는 모든 페이지 복사본에 대해 다음과 같은 방식으로 카운터 값을 조정한다.

- 복사한 이후로 현재까지 일정 비율 이상으로 이득을 본 페이지(예를들어, 페이지 g)의 경우:
  페이지 g를 아직 원격으로 참조하고 있는 처리기들에게 처리기간 인터럽트(interprocessor interrupt)를 보내 페이지 g의 지연 카운터 값을 하향 조정하라고 알림(이는 다른 노드에서 페이지 g에 대한 복사 시기를 앞당기는 효과를 낳는다).
- 복사한 이후로 현재까지 일정 비율 이상으로 손해를 본 페이지(예를들어, 페이지 l)의 경우:
  페이지 l을 아직 원격으로 참조하고 있는 처리기들에게 처리기간 인터럽트를 보내 페이지 l의 지연 카운터 값을 상향 조정하라고 알림(이는 다른 노드에서 페이지 l에 대한 복사 시기를 지연시키는 효과를 낳는다).

위의 카운터 초기값 조정 방식에서 일정 비율 이상으로 이득을 보거나 손해를 본 경우에만 조정하는 이유는 처리기간 인터럽트의 빈도를 조정하기 위한 것으

로 잦은 처리기간 인터럽트는 오히려 추가적인 오버헤드를 유발할 가능성이 있기 때문이다.

1. 각 페이지 복사본에 대한 이득/손실 판별

페이지 카운터 조정 디먼 프로세스는 각 처리기에서 주기적으로 실행되며, 자신의 로컬 메모리에 존재하는 페이지 복사본에 대해 지금까지 복사로 인해 이득을 보았는지 아니면 손해를 보았는지를 결정하는 것이 주된 기능이다. 페이지 카운터 조정 디먼은 페이지를 복사해오는데 드는 비용(C_c)과 로컬 대 원격의 참조 시간 비율(1 : R)을 이용하여 로컬 복사본들에 대해 페이지 복사에 들었던 비용을 상쇄할 수 있을 정도의 횟수 이상으로 로컬 참조가 발생했는지를 검사한다. 여기서 C_c와 R은 시스템마다 정적으로 결정될 수 있는 인자들이기 때문에 페이지 카운터 조정 디먼은 다음과 같은 간단한 식으로 로컬 복사본에 대해 이득/손실 여부를 결정할 수 있다[4].

$$C_c + 1 * N < R * N$$

따라서,

$$N > \frac{C_c}{R-1} \tag{1}$$

부등식 (1)은 로컬 메모리로 복사된 페이지는 복사된 이후로 최소한 $\frac{C_c}{R-1}$ 번 이상의 횟수 만큼 로컬하게 참조되어야 이득을 보게 됨을 나타내고 있다. 그러나 부등식 (1)은 단순히 페이지 복사 비용 만을 고려한 것이고, 페이지를 복사한 이후에 일관성 유지를 위해 발생하는 추가적인 갱신(write-update)에 의한 비용은 고려하지 않은 것이다. 페이지가 복사된 후 그 페이지에 대한 메모리 쓰기로 인해 갱신이 U번 있었고 1번당 비용이 C_u라면 위의 식은 다음과 같이 확장된다.

$$N > \frac{C_c + U * C_u}{R-1} \tag{2}$$

즉, 로컬 메모리로 복사된 페이지는 복사된 이후로 최소한 $\frac{C_c + U * C_u}{R-1}$ 번 이상의 횟수 만큼 로컬하게 참조되어야 이득을 보게 됨을 나타내고 있다.

페이지 카운터 조정 디먼은 로컬 메모리에 존재하는 페이지 복사본에 대한 실제의 참조 횟수(위의 부등식들에서 N에 해당)와 부등식 (2)의 우변값을 비교하여 이득/손실 여부를 결정하고 또한 그 차이를 이용하여 이득 또는 손실의 정도를 나타내는 이득률/손실률을 계산한다. 여기서 이득률은 그 페이지를 로컬 메모리로 복사하지 않고 원격으로 참조했을 때보다 얼마나 메모리 참조 시간이 짧아졌는지를 나타내며, 손실률은 그 반대의 경우를 나타낸다. 이 이득률/손실률을 근거로 페이지 카운터 조정 디먼은 그 페이지의 카운터 값 조정 여부와 카운터 값 증감폭을 결정한다. 그림 1의 수도(pseudo) 코드는 어떤 로컬 복사본 r에 대한 이득/손실 판별 알고리즘을 보이고 있다.

```
if ( N_r > (C_c + U_r*C_u)/(R-1) ) {  /* 페이지 r의 복사로 인해 성능
                                         이득을 본 경우! */
    N_r 과 (C_c + U_r*C_u)/(R-1) 의 차를 이용하여 이득률 계산;
    이득률이 0.1 미만이면 goto do_nothing;
    이득률을 근거로 카운터 값 감소폭을 결정;
    페이지 r을 아직 원격으로 참조하고 있는 처리기들에게
    처리기간
        인터럽트를 보내 결정된 감소량 만큼 카운터 값을
        감소시키라고 지시;
} else if ( N_r < (C_c + U_r*C_u)/(R-1) ) {  /* 페이지 r의 복사로 인해
                                                성능 손실을 본 경우! */
    N_r 과 (C_c + U_r*C_u)/(R-1) 의 차를 이용하여 손실률 계산;
    손실률이 0.1 미만이면 goto do_nothing;
    손실률을 근거로 카운터 값 증가폭을 결정;
    페이지 r을 아직 원격으로 참조하고 있는 처리기들에게
    처리기간 인터럽트를 보내 결정된 증가량 만큼 카운터
    값을 증가시키라고 지시;
} else
do_nothing:
        do nothing;
```

그림 1. 로컬 복사본에 대한 이득/손실 판별 알고리즘
Fig. 1. Performance gain/loss evaluation algorithm for a local copy.

그림 1의 알고리즘에서 이득률/손실률은 다음과 같은 식에 의해 계산된다(계산 결과가 음수인 경우 손실을 의미함).

$$\text{이득률/손실률} = \frac{N_r - \frac{C_c + U_r * C_u}{R-1}}{\frac{C_c + U_r * C_u}{R-1}} \tag{3}$$

한편 카운터 값 증감폭은 식 (3)에 의해 계산된 이득률 또는 손실률의 결과에 따라 결정되는데 이득 또는 손실의 정도에 비례하여 결정된다. 증감폭의 디폴트 값은 250으로 하였으며 이득률/손실률 0.1 마다 증감폭을 50씩 증감하였다. 그리고 증감폭의 최대값은 500으로 설정하였다. 카운터 값 증감폭 결정 과정을 식으로 나

타내면 다음과 같다.

카운터 값 증감폭

$$= \mathrm{MIN}\left(500, \left(250 + |\text{이득률 또는 손실률} \times \left(\frac{50}{0.1}\right)|\right)\right) \quad (4)$$

2. 지연 카운터 값의 동적 조정

페이지 카운터 조정 디먼에 의해 발생된 처리기간 인터럽트를 받은 각 처리기에서는 해당 페이지에 대해 지연 카운터 값을 지시받은 양 만큼 증가시키거나 감소시킨다. 한편 우리는 한 페이지의 지연 카운터 값이 과도하게 증가되거나 감소되는 것을 방지하기 위해 카운터의 상한값과 하한값을 5000과 100으로 각각 설정하였다. 그리고 페이지 카운터 조정 디먼 프로세스는 3초마다 한번씩 각 처리기에서 실행되도록 하였다.

한편 지연 카운터 조정시 동일한 원격 페이지에 대해 어떤 노드로부터는 하향 지시를, 그리고 다른 어떤 노드로부터는 상향 지시를 (거의 동시에)받는 경우가 발생할 수 있는데, 이같은 경우는 페이지 복사후 이득을 본 노드와 손해를 본 노드들이 공존할 수 있기 때문에 발생한다. 특히 같은 페이지에 대한 카운터 값 증감폭이 동일할 경우에는 증감의 효과가 없을 뿐만아니라 오히려 불필요한 카운터값 증감으로 인한 오버헤드만 발생할 수도 있다. 우리는 이러한 경우를 최대한 방지하기 위해 카운터 값 증감 지시를 일정양 만큼 모아두었다가 한꺼번에 조정하도록 하는 방식을 사용하였다. 시스템 내에 총 N개의 처리기가 있는 경우 큐(queue)의 형태로 한 노드에서 최대로 $(N-1)*2$ 개까지 모아둘 수 있도록 하였으며(각 다른 노드 당 평균 2개) 이 큐가 모두 채워졌을 때 한꺼번에 카운터 값을 조정하였다. 따라서 큐에 들어온 카운터 값 조정 지시 중 같은 페이지에 대한 처리는 하나로 모아져 처리될 수 있다. 또한 큐가 채워지지 않은 상태가 상당 기간 지속될 경우 카운터 값 조정도 아울러 지연될 수 있기 때문에 적어도 3초에 한번씩은 큐에 들어와 있는 조정 지시를 처리하도록 하였다. 그림 2의 알고리즘은 다른 노드로부터 카운터 값 조정 지시를 받았을 때의 해당 카운터 값 조정 과정을 보이고 있다.

3. 로컬 복사본에 대한 로컬 참조 횟수 카운트

우리는 로컬 복사본에 대한 로컬 참조 횟수(부등식 (1)(2)에서 N)를 얻기 위해 지연 카운터의 기능을 확장하였다. 일단 어떤 원격 페이지가 지연 카운터에 의한 지연 후에 어떤 처리기의 로컬 메모리로 복사되면 그 페이지의 지연 카운터는 필요없게 된다는 점에 착안하여, 필요없게 된 지연 카운터(다시말해 이미 복사된 페이지의 지연 카운터)가 그 페이지에 대한 로컬 참조 횟수를 세는데 이용될 수 있도록 하였다. 이러한 기능 확장은 N을 세기 위해 별도의 카운터를 설치하는 것 보다 매우 적은 오버헤드를 필요로하므로 효율적이라고 할 수 있으며, 본 논문에서 제시한 기법이나 갱신 카운터 기법에서는 필수적인 기능이라고 할 수 있다.

```
adjust_delay_counter(페이지번호, 증감분)
{
    if (증감 지시 큐가 full이 아니면)
        증감 지시 큐에 현재 요청(페이지번호,증감분)을 첨가;
    else {
        for (증감 지시 큐에 들어 있는 모든 증감 지시에 대해) {
            해당 페이지의 카운터 값을 증감분 만큼 조정;
                (카운터 값이 상한값 또는 하한값을 벗어나는지 검사)
                (같은 페이지에 대한 조정 지시가 여러번 존재할 경우 한꺼번에 조정)
        }
        현재 요청을 증감 지시 큐에 첨가;
    }
}
```

그림 2. 중복 조정을 피하기 위한 지연 카운터 값 조정 과정

Fig. 2. Algorithm for adjusting delay counter value to avoid redundant adjustments.

Ⅳ. 지연 카운터의 구현 방안

본 논문에서 제시하고 있는 "조정가능한 지연 카운터"는 기존의 지연 카운터의 운용을 확장한 것이라고 할 수 있다. 그러나 Bolosky 등[11]의 논문에서는 자신들이 제시한 지연 카운터의 구현 방안에 대해서는 자세히 언급하고 있지 않기 때문에 이번 절에서는 지연 카운터의 구현 방안을 제시한다. 본 논문에서는 구현 방안 제시를 위한 기본 환경으로 RISC(Reduced Instruction Set Computer) 구조의 대표적인 처리기인 MIPS R4000[2]으로 구성된 "버스-기반 공유메모리 NUMA 다중처리기" 시스템을 선정하였다. MIPS R4000을 택한 이유는 MIPS R4000의 메모리 관리 시

2) MIPS R4000은 MIPS Technologies 사의 trademark 임.

스템이 각 기능별로 모듈화가 되어 있고 특히 주소변환을 위한 CPU 상의 TLB(Translation Lookaside Buffer)와 페이지 테이블(메모리 상주)이 분리되어 있어 하드웨어적인 수정을 최소화할 수 있기 때문이다.

먼저 MIPS R4000에서의 개략적인 주소변환 과정을 살펴보면 다음과 같다:

1. 가상 주소에 포함되어 있는 가상 페이지 번호(virtual page number)에 해당하는 주소 변환 정보가 TLB에 존재하는지를 검사.
2. 존재하는 경우(TLB 히트): TLB의 해당 항에서 물리 페이지 프레임 번호를 얻어와 가상 주소에 포함되어 있던 옵셋과 합쳐 물리 주소를 형성한다.
3. 존재하지 않는 경우: TLB 미스(miss)가 발생하여 운영체제로 제어가 넘어간다[3]. 운영체제의 TLB 미스 핸들러는 기존 TLB 항들 중 하나를 제거한 후 페이지 테이블에서 필요로하는 항을 찾아 TLB를 채운 후 역시 같은 방식으로 물리 주소를 형성한다.

위의 과정에서 알 수 있는 것은 MIPS R4000에서는 페이지 테이블의 구조나 위치등이 운영체제에 의해 소프트웨어적으로 관리된다는 점이다. MIPS R4000은 오직 TLB 만을 참조하여 가상 주소를 물리 주소로 변환한다. 따라서 지연 카운터 구현을 위한 하드웨어 수정은 CPU 내의 TLB 구조에 카운터 필드 추가와 이 필드에 대한 하향/상향 카운팅 로직, 그리고 카운터 값이 0이 되었을 때 CPU에 인터럽트를 발생시키는 기능 추가등으로 요약될 수 있다. 운영체제 측면에서는 페이지 테이블 구조에 카운터 필드가 첨가되어야 한다는 점과 TLB와 페이지 테이블의 카운터 필드에 대한 초기화 및 값 조정 기능을 지원해야 한다는 점 등이 추가되어야할 사항에 속한다.

1. TLB 엔트리 구조

그림 3은 MIPS R4000이 32 비트 모드일 때 사용하는 TLB의 한 엔트리 구조를 보이고 있다(각 필드의 자세한 의미는 참고 문헌 [5] 참조).

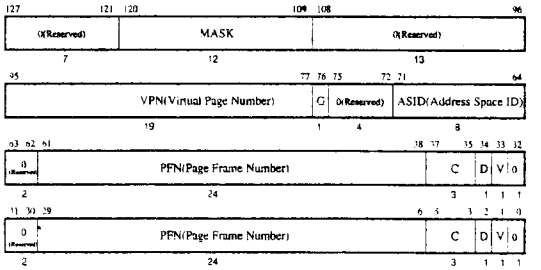


그림 3. MIPS R4000의 TLB 엔트리의 구조(32-비트 모드 일 때): 총 128 비트로 구성

Fig. 3. TLB entry structure of MIPS R4000 processor(32-bit mode): The total size is 128 bits.

우리는 충분한 크기의 카운터 값을 지원하기 위해 카운터 필드로 13비트(카운터 값 8192까지 가능)를 설정하였고, 원격 페이지와 로컬 페이지를 구분하기 하기 위해 1비트를 더 설정하였다. 원격 페이지인 경우는 참조시 하향 카운팅을, 그리고 로컬 페이지인 경우는 참조시 상향 카운팅을 해야 하므로 원격/로컬 페이지 구별 필드는 필수적이다(카운터 필드의 상향/하향 카운팅 로직은 CPU 내에 구현 가능하다고 가정함). 또한 어떤 페이지의 카운터 값이 0이 되었을 때에도 원격/로컬 페이지 구별 비트가 이용되는데 이 비트가 "원격"인 경우에만 인터럽트가 발생해야 한다.

이처럼 본 논문에서 제시한 카운터를 구현하기 위해 필요한 비트 수는 TLB의 한 엔트리당 14 비트인데 그림 3을 보면 0으로 표시되어 있는 예약 비트[4]가(예를 들면, 비트 96 ~ 108, 비트 121 ~ 127) 충분하다는 것을 알 수 있다. 이 예약 비트 필드들을 이용하면 TLB 엔트리에 별도의 비트를 두지 않아도 카운터 필드와 원격/로컬 페이지 구별 필드를 확보할 수 있다.

2. 운영체제 지원 사항

TLB 엔트리와 마찬가지로 운영체제에서도 페이지 테이블 구조에 카운터 필드와 원격/로컬 페이지 구별 필드를 첨가해야 한다. 카운터 필드의 초기값은 페이지 복사 정책의 성능에 큰 영향을 미치기 때문에 성능 평가시 다양한 값으로 변화시켜가며 실험하였다(5장 참

3) MIPS R4000에서 TLB 교체 정책은 하드웨어에 의존할 수도 있고 운영체제에 의존할 수도 있다. 하드웨어에 의존할 경우는 임의(random) 교체 정책 만이 지원된다. 반면 운영체제에 의존할 경우는 하드웨어에 의존할 경우 보다 성능는 떨어지지만 다양한 정책을 구현할 수 있다. 여기서는 TLB 교체 정책이 운영체제에 의해 수행되는 경우를 가정한다.

4) 일반적으로 CPU 내의 시스템 레지스터 필드 중 예약 비트는 차후의 CPU 설계에 사용될 것을 예상하여 확보해놓은 비트들을 의미한다.

조). 원격/로컬 페이지 구별 비트는 응용 프로세스 실행 초기화 시에는 모든 사용자 페이지에 대해 "원격"으로 지정되며(물론 실행 초기화 시에 이미 로컬 메모리에 존재하는 페이지는 "로컬"로 지정된다), 이후에 카운터 값이 0이 되어 로컬 메모리로 복사된 페이지에 대해서는 "로컬"로 지정된다.

TLB 미스 핸들러의 기능에 추가 되어야 할 사항은 TLB 교체 정책(replacement policy)에 의해 교체당하는 TLB 엔트리에 있는 카운터 값과 원격/로컬 구별 필드를 해당 페이지 테이블에 기록해 두어야 한다는 점이다. 교체된 TLB 엔트리가 후에 다시 TLB로 적재되었을 때 과거 교체될 당시의 값들을 가지고 있어야 하기 때문이다.

처리기간 인터럽트를 통해 다른 노드의 페이지 카운터 조정 디먼으로부터 카운터 값 조정 지시를 받으면 처리기간 인터럽트 핸들러가 실행되는데 이 핸들러에서는 페이지 번호와 조정폭을 인자로 받아 페이지 번호에 해당하는 페이지를 TLB와 페이지 테이블에서 찾아 카운터 값을 조정해주면 된다.

V. 성능 평가 및 결과

이번 절에서는 본 논문에서 성능 실험을 위해 사용한 시뮬레이션 환경에 대해 기술하고, 시뮬레이션에 사용된 시스템 인수(system parameter)들과 병렬 응용 프로그램들에 대해 설명한다.

1. 시뮬레이션 환경

우리는 최대 32개의 처리기를 가진 NUMA 공유 메모리 다중처리기를 시뮬레이션하기 위해 실행-구동형(execution-driven) 시뮬레이션 기법을 이용하였다. 사용된 시뮬레이터는 크게 전위(front end) 시뮬레이터와 후위(back end) 시뮬레이터로 구성되어 있는데(그림 4, 전위 시뮬레이터는 입력된 병렬 응용 프로그램의 실행 코드(object code)를 해석(interpret)하여 각 처리기들의 실행을 시뮬레이션하는 기능을 한다. 우리는 전위 시뮬레이터로 MINT(Mips INTerpreter)[16][17][18]를 이용하였다. MINT는 현재 MIPS II 인스트럭션 세트(instruction set) 만을 지원한다. 후위 시뮬레이터는 MINT의 출력을 바탕으로 메모리 시스템을 시뮬레이션 한다.

MINT는 응용 프로그램의 실행 코드를 해석하여 매 메모리 참조[5]시마다 후위 시뮬레이터에서 제공한 일정 함수들을 호출하기 때문에 시뮬레이션하고자 하는 페이지 복사 정책은 실제적으로 후위 시뮬레이터에 의해 구현된다.

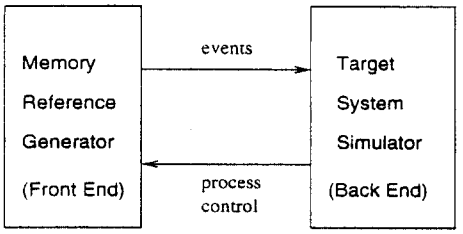


그림 4. 성능 평가에 사용된 시뮬레이터의 구성
Fig. 4. Major components of the simulator used for performance evaluation.

MINT의 경우 우리는 미국 로체스터 대학의 것을 수정없이 사용하였고[6], 후위 시뮬레이터의 경우는 본 논문에서 제안한 "조정가능한 지연 카운터" 메커니즘을 NUMA 공유 메모리 다중처리기 시스템 상의 페이지 복사 정책에 적용할 수 있도록 새로 구현하였다.

2. 실험에 쓰인 인수 및 병렬 응용 프로그램

표 1은 우리가 시뮬레이션 시에 사용한 시스템 인수들을 보이고 있다. 이 인수들은 주로 메모리 참조 비용 등과 같이 페이지 복사 정책 전반에 걸쳐 발생하는 여러 하드웨어 비용들을 시뮬레이션에 반영하기 위한 것이다[12].

표 1. 실험에 이용된 하드웨어 비용 관련 인수들
Table 1. Hardware cost-related parameters used in simulation.

하드웨어 비용 관련 인자	비용(단위:cycle)
로컬 메모리 참조	5
원격 메모리 읽기	100
원격 메모리 쓰기	10
네트워크 지연	90
페이지 크기	4K
복사를 위한 페이지 전송 비용	4096
페이지 부재 폴트 비용	500
갱신(Write-Update) 비용	10

5) 메모리 참조는 데이타 참조를 뜻하는 것이며 instruction fetch를 위한 메모리 참조는 제외됨
6) ftp로 다운로드 받아 사용하였음

표 1에서 **네트워크 지연**(network latency)는 하나의 메시지가 처리기들 간의 상호 연결망을 왕복(round-trip)하는데 드는 비용을 의미한다. **원격 메모리 읽기**는 **네트워크 지연** 비용과 **로컬 메모리 참조** 비용, 그리고 기타 약간의 하드웨어 비용을 합한 것이다. **원격 메모리 쓰기** 비용은 **로컬 메모리 참조** 보다는 약간 크지만 **원격 메모리 읽기** 보다는 상당히 적은데, 쓰기 연산은 읽기 연산과는 달리 처리기를 중단(stall)시킬 필요가 없기 때문이다. 메모리 일관성을 위해 발생하는 **갱신**의 비용은 **원격 메모리 쓰기** 비용과 비슷하게 설정하였다.

본 시뮬레이션에서는 미국 스탠포드 대학의 SPLASH 사이트[9]에서 얻을 수 있는 병렬 응용 프로그램중 4개의 프로그램을 수행하였다. 본 시뮬레이션에 이용된 병렬 응용 프로그램의 종류와 특성은 표 2와 같다.

표 2. 시뮬레이션에 이용된 병렬 응용 프로그램의 종류 및 특성

Table 2. The parallel applications used in simulation and their characteristics.

응용 프로그램의 이름	총 메모리 참조횟수 (단위: 10^6)	Working Set 크기 (단위: MB)	응용 프로그램의 기능
MP3D	8.935	2.02	simulate rarefied hypersonic flow
BARNES	58.716	1.20	simulate evolution of galaxies
CHOLESKY	39.341	2.88	Cholesky factorize a sparse matrix
RADIX	33.107	3.76	Parallel integer radix sort on Connection Machine CM-2

한편 표 3은 32개의 처리기를 이용하였을 때 각 응용 프로그램들의 네가지 메모리 참조 패턴을 전체 메모리 참조 횟수에 대한 비율의 형태로 보이고 있다. 표 3에서 제일 오른쪽 컬럼은 공유 쓰기(shared write) 횟수와 공유 읽기(shared read) 횟수의 비율을 보이고 있다.

3. 실험 결과

우리는 본 논문에서 제시한 메커니즘의 성능과 기존의 고정형 지연 카운터 메커니즘의 성능을 비교하기 위해 *NAMC*(Normalized Average Memory access Cost)[3]를 성능 측정의 척도(metric)로 사용하였다. *NAMC*는 1번의 메모리 참조에 있어서 로컬 메모리 참조 비용에 대한 평균 비용을 뜻하는데, 예를들어 *NAMC*가 2라는 것은 평균 메모리 참조 비용이 이상적인 경우(모든 메모리 참조가 로컬 메모리에서 이루어질 경우)에 비해 2배 크다는 것을 의미한다.

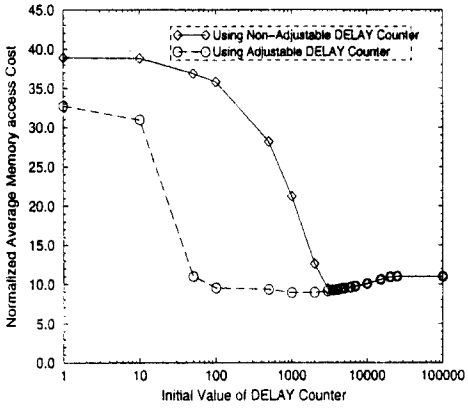
표 3. 각 병렬 응용 프로그램들의 메모리 참조 패턴(32개의 처리기 사용)

Table 3. Memory reference pattern of applications used in simulation(using 32 processors.

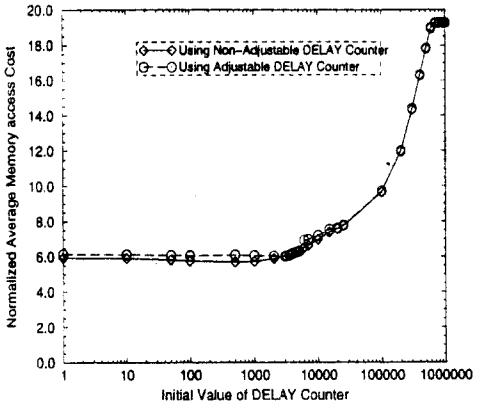
응용 프로그램의 이름	Private Read(%)	Private Write(%)	Shared Read(%)	Shared Write(%)	Shared Write : Shared Read
MP3D	37.6	20.0	24.5	17.9	0.73 : 1
BARNES	49.7	36.2	13.5	0.6	0.04 : 1
CHOLESKY	25.0	8.6	57.2	9.2	0.16 : 1
RADIX	49.9	25.6	15.1	9.4	0.62 : 1

그림 5의 각 그래프들은 사용된 각 응용 프로그램에 대해 기존의 고정형 지연 카운터 방식과 우리가 제시한 조정가능한 지연 카운터 방식의 성능을 *NAMC*로 표현하고 있다. 각 그래프의 x 축은 지연 카운터 초기값의 변화를, y 축은 이에 따른 *NAMC* 값의 변화를 보이고 있다. 그래프에서 x 축의 제일 왼쪽 값인 1은 매 첫 원격 참조때 페이지를 복사해오는(immediate replication) 정책을 의미하고, x 축의 제일 오른쪽 값인 100,000이나 1,000,000은 원격으로 100,000번 혹은 1,000,000번 참조한 후에 페이지를 복사해오는 정책을 의미하는데 이는 페이지 복사를 하지 않는 정책과 같다.

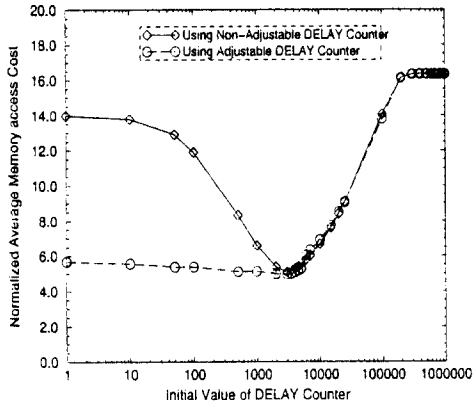
그림 5의 각 그래프들이 보이고 있는 의미는 첫째로는 응용 프로그램마다 약간의 차이는 있으나 본 논문에서 제시한 기법이 사용된 응용 프로그램의 대부분에 걸쳐 전반적으로 좋은 성능을 나타낸다는 것이다. 둘째로는 지연 카운터 초기값의 변화에 따른 *NAMC*의 변화가 심하지 않고 고른 성능을 나타냄을 알 수 있다. 예를들어 MP3D의 경우 고정형 지연 카운터 방식을 이용하면 최적의 성능을 나타내는 지연 카운터 초기값의 범위가 대략 3000 ~ 5000으로 매우 좁은 것으로 나타났지만 우리의 방식을 이용할 경우에는 100 ~ 5000으로 상당히 넓은 범위에 걸쳐있음을 알 수 있었다. CHOLESKY의 경우는 고정형 지연 카운터를 이용하면 2000 ~ 5000에서 최적의 성능을 보이고 있으나 조정가능한 지연 카운터를 이용할 경우 1 ~ 5000 사이에서 거의 최적에 가까운 최적의 성능을 보이고



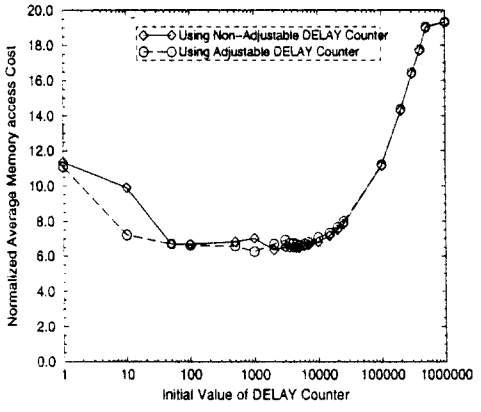
(a) MP3D



(b) BARNES



(c) CHOLESKY



(d) RADIX

그림 5. 고정형 지연 카운터와 조정가능한 지연 카운터를 이용했을 때의 성능 비교 결과
Fig. 5. The performance comparison of the two mechanisms(when using non-adjustable *DELAY* counter and adjustable *DELAY* counter).

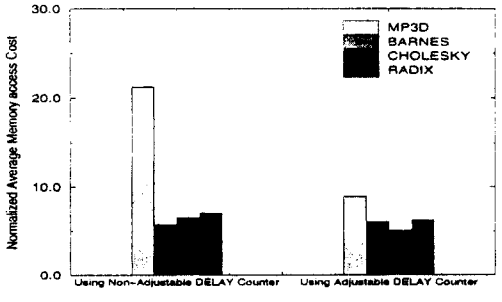


그림 6. 지연 카운터 초기값이 1000일 때 각 응용 프로그램의 성능 비교
Fig. 6. Performance comparion of the two mechanisms when the initial value is 1000.

있다. 셋째는 기존의 방식을 이용할 경우 응용 프로그램마다 최적의 성능을 나타내는 지연 카운터 초기값이 큰 차이를 보이기 때문에 구현시 어떤 값을 선택할 것이냐 하는 문제가 야기될 수 있으나 우리의 방식에서는 최적의(또는 최적에 가까운) 성능을 보이는 지연 카운터 초기값의 범위가 넓고 응용 프로그램마다 큰 차이를 보이지 않기 때문에 지연 카운터 초기값 선정에 있어서 유연성(flexibility)을 높일 수 있는 장점이 있다. 그림 6은 지연 카운터 초기값이 1000일 때 각 응용 프로그램들의 *NAMC*를 보이고 있는데 조정가능한 지연 카운터 방식이 고정형 지연 카운터 방식에 비해 좋고 고른 성능을 보인다는 것을 알 수 있다. BARNES의 경우는 고정형 지연 카운터를 이용한 경우와

조정가능한 지연 카운터를 이용했을 때의 성능이 거의 차이를 보이고 있지 않은데 이것은 BARNES의 다소 예외적인 메모리 참조 패턴에 기인한다.

표 3에서 BARNES의 경우 공유 읽기와 공유 쓰기의 횟수가 전체 메모리 참조 횟수에서 차지하는 비율이 다른 응용 프로그램에 비해 상당히 낮음을 알 수 있고, 특히 공유 쓰기의 비율이 매우 낮음을 알 수 있다. 이로 인해 조정가능한 지연 카운터 방식에 의해 이득을 보는 경우가 다른 응용 프로그램에 비해 다소 적었다는 것을 알 수 있으며 이같은 메모리 참조 패턴의 응용 프로그램에 대해서는 기존의 고정형 지연 카운터 방식이나 즉각적인 복사(immediate replication) 정책만으로도 충분한 성능을 얻을 수 있다는 것을 알 수 있다.

VI. 결 론

NUMA 구조의 공유 메모리 다중처리기 시스템에서 참조 국지성의 활용을 극대화하기 위해 본 논문에서는 "조정가능한 지연 카운터"를 이용한 페이지 복사 정책을 제시하였다. 이 메커니즘의 효율성을 평가하기 위해 우리는 실행-구동형 시뮬레이션 기법을 이용하였으며, 성능 평가를 통해 조정가능한 지연 카운터의 사용이 기존의 고정형 지연 카운터 기법에 비해 병렬 응용 프로그램들의 다양한 페이지 참조 패턴에 좀 더 나은 적응력을 보인다는 것을 알 수 있었다. 실험에서 얻은 주요한 결론들은 다음과 같다.

조정가능한 지연 카운터를 이용할 경우 기존의 고정형 지연 카운터에 비해 보다 넓은 범위의 카운터 초기값에 걸쳐서 좋은 성능을 나타낸다. 또한 응용 프로그램마다 메모리 참조 패턴이 차이를 보인다 하더라도 조정가능한 지연 카운터에 의해 페이지 복사 정책이 다양한 페이지 참조 패턴에 적응할 수 있다. 따라서 각 응용 프로그램들이 고른 성능을 보인다.

그리고 최적의(또는 최적에 가까운) 성능을 나타내는 카운터 초기값의 범위가 기존의 것 보다 상당히 넓어졌기 때문에 카운터 초기값 선정이 용이해졌을 뿐만아니라 잘못된 초기값 선정으로 인해 시스템의 성능이 저하될 가능성이 낮아졌다. 이로 인하여 운영체제의 메모리 관리 정책의 융통성(flexibility)과 적응력(adaptability)을 높이고 참조 국지성의 이용을 극대화할 수 있다.

참 고 문 헌

[1] William J. Bolosky, Michael L. Scott, Robert P. Fitzgerald, Robert J. Fowler, and Alan L. Cox. Numa Policies and Their Relation to Memory Architecture. In *Proceedings of the 4th International Conference on Architectural Support for Programming Language and Operating Systems*, pages 212-221, April 1991.

[2] Evangelos P. Markatos and Catherine E. Chronaki. Trace-Driven Simulation of Data-Alignment and other Factors affecting Update and Invalidate Based Coherent Memory. In *Proceedings of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems(MASCOTS '94)*, pages 44-52, January 1994.

[3] Evangelos P. Markatos and Catherine E. Chronaki. Using reference counters in Update Based Coherent Memory. Technical report, University of Rochester, Department of Computer Science, Rochester, NY 14627-0226. Also appeared in PARLE '94(Parallel ARchitectures and Languages Europe), July 1994.

[4] R. P. LaRowe jr., C. S. Ellis, and L. S. Kaplan. The robustness of NUMA memory management. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, pages 137-151, Octber 1991.

[5] Joe Heinrich. *MIPS R4000 Microprocessor User's Manual*, chapter 4, pages 61-97. MIPS Technologies, Inc., 2nd edition, 1994.

[6] J. E. Veenstra. Mint Tutorial and User Manual. Technical report, TR452, Computer Science Department, University of Rochester, July 1993.

[7] J. E. Veenstra and R. J. Fowler. Mint: A Front End for Efficient Simulation of Shared-Memory Multiprocessor. In *Proceedings of the Second International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Sys-*

tems(MASCOTS '94), pages 201-207, January-February 1994.

[8] M. Marchetti, L. I. Kontothanassis, R. Bianchini, and M. L. Sc ott. Using Simple Page Placement Policies to Reduce the Cost of Cache Fills in Coherent Shared-Memory System. Technical report, TR535, University of Rochester, Department of Computer Science, September 1994.

[9] J. P. Singh, W. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. *ACM SIGARCH Computer Architecture News*, 20(1):5-44, March 1992.

저 자 소 개

李 鍾 雨(正會員)

1967년 11월 4일생. 1990년 서울대학교 컴퓨터공학과 졸업(학사). 1992년 서울대학교 컴퓨터공학과 대학원 석사과정 졸업(석사) 1994년 서울대학교 컴퓨터공학과 박사과정 수료. 1994년 ~ 현재 : 현대전자 소프트웨어 연구소 연구원 주 관심분야 : 운영체제, 분산 시스템, 병렬 운영체제 등

曺 裕 根(正會員)

1949년 3월 24일생. 1971년 서울대학교 공과대학 건축공학과 졸업 1973년 서울대학교 공과대학 건축공학과 석사. 1978년 미국 미네소타대학 전산학과 박사학위 취득 1979년 ~ 현재 : 서울대학교 컴퓨터공학과 교수. 1984년 ~ 1985년 : 미국 미네소타대학 교환 교수 1993년 ~ 1995년 : 서울대학교 중앙교육연구전산원장 1995년 : 한국정보과학회 부회장. 주 관심분야 : 운영체제, 알고리즘 설계와 분석