S/W 관리 TLB의 초기접근실패 감소 기법

박 장 석†

요 약

본 논문에서는 S/W 관리 TLB(Translation Lookaside Buffer)의 실패 패널티를 감소시키기 위하여 사용될 TLB 엔트리를 선인출하여 S/W 관리 TLB의 초기 접근 실패를 감소시키는 새로운 기법을 제시한다. 이 방법은 특별한 응용프로그램에만 동작하는 것은 아니다. 제안하는 방법의 핵심은 임의의 페이지에 대한 첫번째의 TLB 접근 전에 선인출 명령어를 수행하여 TLB 실패를 방지하는 것이다. 선인출할 페이지를 예측하기 위하여 명령어의 수행 관점에 기본을 둔 새로운 TLB 실패 분류법을 제안하고, 이를 이용한 알고리즘과 구현방법을 기술하며, 정량적인 분석에 의해 제안한 기법이 S/W 관리 TLB의 성능 향상을 위한 유효한 기법임을 보인다. 또 제안한 방법이 S/W 관리 TLB에서 버스 트래픽을 감소시키는 부가적인 장점에 대해 논한다.

The Reducing Technique of Compulsory Misses for S/W managed TLB

Jang Suk Park†

ABSTRACT

This paper introduces a new technique for reducing the compulsory misses of software-managed TLBs by prefetching necessary TLB entries before being used. This technique is not inherently limited to specific applications. The key of this scheme is to perform the prefetch operations to update the TLB entries before first accesses so that TLB misses can be avoided. For the identifications of the prefetch pages, the new classification is introduced, which is based on the view of an object code execution. Then, the algorithms and the implementation technique are described. Using a quantitative analysis, the proposed scheme is evaluated to prove that it is a useful technique for the performance enhancement of the S/W managed TLBs. In addition, it is discussed that reducing the miss rate by the prefetch scheme reduces the total miss penalty and bus traffics in S/W-managed TLBs.

1. 서 론

TLB는 가상주소 번역에서 유용한 자원임이 입증되어 왔으나, 주기억 장치 접근 시간을 줄이기 위한 계층적 기억 장치(hierarchical memory system)에 대한 연구는 주로 캐시기억장치에 국한되어 왔고 TLB에 대한 연구는 그 중요성에 비해 많지 않다. Clark와 Emer[1]는 VAX-11/780을 제어하여 H/W 측정과 시뮬레이션에 의해 H/W 관리 TLB의 여러 특성을 측정하고 평가하였다.

Chen[2]은 SPEC 벤치마크로 부터 얻은 추적 자료(trace data)를 사용한 시뮬레이션 기반 연구를 통하여 여러 가지 TLB 구조의 성능을 조사하였다. 그들

† 정 회 원: 정보통신연구관리단 정보기술평가1실
논문접수: 1997년 9월 25일, 심사완료: 1997년 12월 6일

은 2단계 TLB와 완전 연관 TLB(fully-associative TLB)의 성능을 평가하고, TLB 성능에서 가장 중요한 요인은 사상되는 기억장치의 정도임을 보였다.

Talluri[3]는 두개의 페이지 크기를 지원하여 TLB 실패를 감소시키는 방법에 대해 조사하였다. 일반적으로 페이지 크기를 증가시키면 실패율은 감소되나 작업 세트(working set)가 상대적으로 증가되어 기억장치의 낭비를 초래하는 단점이 있다. 그러나, 그들은 4KB와 32KB의 두개의 페이지 크기를 지원하여 작업 세트 크기가 많이 증가하지 않으면서도 TLB 성능을 개선시킬 수 있음을 보였다.

Nagle[4, 5]은 최초로 S/W 관리 TLB의 성능 문제에 대해 연구하였다. Nagle은 S/W 관리 TLB의 장단점을 조사하고, S/W 관리 TLB와 통합 운영체제 및 마이크로커널 기반 운영체제의 상호 관계에 대해 조사하였다. 그들은 동일한 응용프로그램에 대해 전체 TLB 서비스 시간은 운영체제에 따라 상당한 차이가 있음을 보였다. 그리고, S/W 관리 TLB의 성능을 개선시키기 위한 H/W 기반의 기법들을 제시하였다.

그러나, 이와같은 이전의 연구들은 TLB 파라미터 변경에 따른 성능 평가 위주의 연구 결과이며, 이전 연구로 제안된 H/W 기반의 방법들로 S/W 관리 TLB에 적용될 수 있는 성능 향상 방법을 요약하면 다음과 같다. 첫째 방법은 TLB 엔트리 수를 증가시키거나 또는 연관성(associativity)의 정도를 증가시키는 방법이다[1, 2, 4, 5]. 이 방법은 TLB 접근 시의 성공율은 개선시키나, TLB 파라미터를 증가시키는 것은 추가 비용이 요구되고, 프로세서 다이에서 더 넓은 부분을 차지하게 되어 결과적으로 번역 시간이 더 소비될 수도 있다[3, 6]. 또, 사이클당 다수의 기억 장치 접근을 하는 슈퍼스칼라 시스템에서는 TLB가 다중 포트(multi-port)를 지원하여야 하므로, 만약 TLB가 너무 커지면 모든 기억 장치의 접근에 부정적 영향을 줄 수 있다[3]. 둘째 방법은 시스템의 페이지 크기를 증가시키는 방법이다[7]. TLB 엔트리 수가 일정할 때, 시스템의 페이지 크기가 증가하면 TLB의 성공율은 증가된다. 그러나, 이 방법은 가상 기억 장치에서 내부 단편화(internal fragmentation) 및 외부 단편화(external fragmentation)를 초래하기 때문에 단순한 방법은 아니다[3, 8]. 셋째 방법은 TLB에 다중 페이지 기법을 구현하는 것이다[3]. MIPS R4000, SUN SuperSPARC, HP PA-RISC 및 Intel i860 XP 등의 프로세서들은 TLB에 다중 페이지 기법을 지원하고 있다. 이 방법은 큰 페이지의 장점을 이용하기 위한 적절한 페이지 할당 정책 선택 방법과 페이지 대체 정책(replacement policy)에서 다중 페이지 크기를 조화시키는 운영체제의 정책을 필요로 한다. 그러나, 현재까지 이 기능을 지원하는 운영체제가 없으며, 이와 같은 [illegible] 해결을 여전히 필요로 하고 있어 현재로서는 미성숙된 기술이다[3].

이와 같은 기법들은 H/W 관리 TLB에도 적용될 수 있는 방법들로, TLB 실패 원인 분석에 이용되는 3C 모델(Compulsory Miss: 초기접근실패, Capacity Miss: 용량접근실패, Conflict Miss: 충돌접근실패)[6, 9, 11]에서 용량접근실패와 충돌접근실패를 주로 감소시키는 방법들이다. 그러나, TLB의 실패율이 높을수록 충돌접근실패 및 용량접근실패의 비율이 높지만, TLB의 실패율이 낮을수록 초기접근실패의 비율은 상대적으로 증가한다[10]. 현재 대부분의 프로세서의 TLB는 높은 연관성을 갖고 많은 엔트리 수를 갖기 때문에 충돌접근실패 및 용량접근실패의 비율이 낮아지고 있는 사실을 고려한다면, 향후의 컴퓨팅 환경에서는 초기접근실패의 비율은 증가할 것으로 전망된다. 그래서, S/W 관리 TLB에서 페이지 번역 과정에서 융통성을 제공하는 장점을 가지면서 H/W 관리 TLB 수준의 성능을 얻기 위해서는 H/W 관리 TLB에서는 감소하기 어려운 초기접근실패 감소는 중요한 이슈가 된다.

본 논문에서는 S/W 관리 TLB를 사용하는 시스템의 성능 향상을 위하여 S/W 관리 TLB의 초기접근실패를 감소시키는 방법을 제시하고자 한다. 본 연구에서 제안하는 S/W 제어 선인출의 기본 개념은 프로그램에 대한 컴파일 시의 지식과 TPU(TLB Prefetch Unit)라고 하는 H/W을 통하여 TLB 엔트리를 선인출함으로써 초기접근실패를 방지하는 것이다. 본 연구의 성능 개선 방법은 두 가지로 요약할 수 있다. 첫째 이득이 선인출 명령어와 TPU H/W을 사용하여 S/W 관리 TLB의 실패 서비스 시에 초래되는 문맥 교환 오버헤드를 제거하는 것이고, 둘째가 TLB 실패 서비스에 소요되는 시간을 고려하여 선인출 명령어를 필요한 시점보다 미리 수행되게 하여 TLB 실패 서비스에 소요되는 시간 자체를 제거 함으로써, TLB 실패

서비스로 인한 목적프로그램의 정지(stall) 시간을 감소키는 것이다.

본 논문의 구성은 2장에서는 선인출할 페이지를 찾기 위한 효율적인 TLB 실패 분류법과 알고리즘 및 동작을 기술하며, 3장에서는 TLB 엔트리를 선인출하기 위하여 TPU를 구동시키는데 이용되는 선인출 명령어와 TPU의 구현 방법에 대해 기술한다. 4장에서는 제안한 TLB 선인출 기법이 유효한 기법임을 정량적으로 분석하고, 결론 및 향후 연구방향을 기술한다.

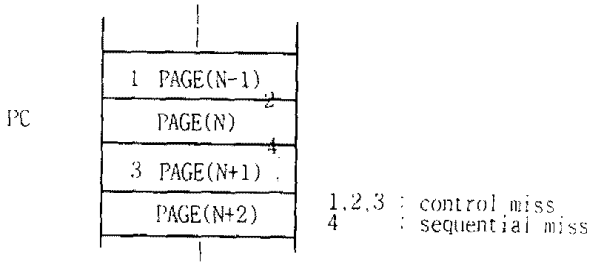
2. 실패 분류

계층적 기억 장치에서 실패의 원인을 분류하는 3C [6, 9, 11] 와는 대조적으로, 제안하는 새로운 분류법은 목적 프로그램의 수행 관점에서 선인출할 페이지를 판단하는 정보를 제공한다. 본 논문에서는 이를 '선인출 후보'라고 정의한다. 제안하는 분류법에서 각 실패는 가상주소에 대한 실 주소 값이 TLB에 저장되어 있지 않으면, TLB 실패가 발생할 가능성이 있는 목적 프로그램에서의 명령어 위치를 말한다. 이 개념은 가상주소 번역을 위하여 TLB를 사용하는 모든 프로그램에 적용할 수 있다. 따라서, 운영체제를 포함하는 모든 시스템 프로그램들도 사상된 주소 방법(mapped address scheme)을 사용하는 경우에 마찬가지로 적용될 수 있다.

만약 주소 공간이 외부 인터럽트에 의해 변경이 되지 않으면, TLB에서 모든 실패는 목적 프로그램 수행 관점에서 보면 다음의 세 가지 중 하나로 해석될 수 있다:

1) 순차 실패(Sequential miss): 명령어 수행에 의해 현재의 페이지가 다음 페이지로 변경될 때 예측되는 실패이다. 만약 명령어 TLB에 이 페이지에 대한 번역 내용이 저장되어 있지 않으면, TLB 실패가 발생된다.
2) 제어 실패(Control miss): 분기 명령어(branch instruction) 또는 제어 명령어 수행에 의해 현재의 페이지가 다른 페이지로 변경될 때 기대되는 TLB 실패이다. 만약 명령어 TLB에 이 페이지에 대한 번역 내용이 저장되어 있지 않으면, TLB 실패가 발생된다.
3) 데이터 실패(Data miss): 데이터 인출을 위하여 현재의 페이지에서 다른 페이지로 접근이 이루어 질 때 기대되는 실패이다. 만약 데이터 TLB에 이 페이지에 대한 번역 내용이 저장되어 있지 않으면 TLB 실패가 발생된다.

위에서 분류한 순차 실패와 제어 실패는 주기억장치로 부터 명령어 인출 시에 발생되는 것이고, 데이터 실패는 오퍼런드 인출 시에 발생되는 것이다. 순차 실패와 제어 실패의 개념적 다이어그램은 (그림 1)과 같다. (그림 1)에서 PC(Program Counter)가 페이지 N을 가리킬 때, 다음에 수행할 명령어는 페이지 N 내부에 있거나 또는 페이지 경계를 벗어나는 경우에는 네 가지 경우로 볼 수 있다. 4번의 경우 정상적인 명령어의 수행 또는 페이지 N+1으로의 분기에 의해 발생될 수 있는 순차 실패를 의미하고, 1, 2, 3의 경우는 분기 명령어 또는 제어 명령어에 의한 제어 실패를 의미한다.



(그림 1) 제어 실패와 순차 실패의 개념적 다이어그램
(Fig. 1) The conceptual diagram of control miss and sequential miss

3. 알고리즘 및 동작

3.1 알고리즘

TLB 실패를 발생시키는 페이지 엔트리를 찾기 위해서는 모든 명령어와 명령어의 오퍼랜드 인출을 위해 접근하는 페이지 순서를 예측할 수 있어야 한다. 임의의 목적 프로그램이 $I_1, I_2, \cdots, I_n$의 명령어 조합(여기서 n은 자연수)으로 구성된다고 할 때 다음과 같이 정의한다.

1. IPS(i): I_i 인출 동안에 TLB을 접근할 가능성이 있는 페이지 집 합을 의미한다. 여기서 $1 \le i \le n$이다.
2. DPS(i): I_i 명령어를 수행하는 동안 데이터 인출을

위하여 TLB을 접근할 가능성이 있는 페이지 집합을 의미한다. 여기서 $1 \le i \le n$이다.

3. PI(i): I_{i+1} 명령어 인출을 위해서 사용한 페이지를 의미한다. 여기서 $1 \le i \le m$이고 $m < n$ 이다.
4. NP(i): 다음 수행할 명령어 인출을 위하여 접근하는 페이지를 의미한다. 여기서 $1 \le i \le m$이고 $m < n$ 이다.
5. PD(i): I_i 명령어를 수행하는 동안 데이터 인출을 위해서 사용된 페이지를 의미한다. 여기서 $1 \le i \le m$이고 $m < n$이다.
6. ND(i): 다음 수행할 명령어의 데이터 인출을 위하여 접근하는 페이지를 의미한다. 여기서 $1 \le i \le m$이고 $m < n$이다.

본 논문에서 제안하는 선인출 기법의 전체적 개념은 I_1에서 I_n까지 모든 I_i, $1 \le i \le n$에 대해 IPS(i)와 DPS(i)을 미리 TLB에 저장이 되게 하는 것이다. 앞에서의 정의로 부터 IPS(i)와 DPS(i)는 (1)과 (2)로 표현될 수 있다.

$$IPS(i) = \{PI(i), NI(i)\} \quad (1)$$

$$DPS(i) = \{PD(i), ND(i)\} \quad (2)$$

수식 (1)과 (2)에서 PI(i)와 PD(i)의 번역값은 항상 TLB에 저장되어 있으나, 만약 NI(i)가 페이지 PI(i)+1과 같고, 이 페이지가 TLB에 저장되어 있지 않으면 순차 실패가 발생된다. 또, NI(i)가 PI(i)+1과 같지 않고 페이지 NI(i)가 TLB에 저장되어 있지 않으면 제어 실패가 발생된다. 이 경우에서 I_i가 분기 명령어 또는 제어 명령어이면 수식 (1)은 (3)과 (4)로 분리될 수 있다.

If $NI(i) \neq PI(i) + 1$,

$$IPS(i) = \{PI(i), NI(i)\} \quad (3)$$

Otherwise,

$$IPS(i) = \{PI(i), PI(i) + 1\} \quad (4)$$

그래서, 수식 (4)는 순차 실패로 간주되고, 수식 (3)만 제어 실패로 고려한다. 또, 수식 (2)의 ND(i)는 데이터 실패를 의미한다. 수식 (2)의 PD(i)와 ND(i)는 복수 개의 페이지를 가질 수 있다. 그 이유는 어떤 명령어들은 복수의 오퍼랜드를 가질 수 있기 때문이고, 접근하는 페이지 수는 명령어 I_i의 오퍼랜드 수에 달려 있다. 따라서, 모든 명령어에서 NI(i)와 명령어 I_i 수행 시에 오퍼랜드 인출(operand fetch)을 위해 필요한 ND(i) 등은 TLB을 최초 접근 시에는 선인출 후보가 될 수 있다.

여기서부터는 앞에서 제시된 분류법을 사용하여 목적 프로그램에서 NI(i)와 ND(i)에 대한 TLB 선인출이 어떻게 이루어지는 지를 설명한다. 기본 개념은 목적 프로그램의 각 실패 위치를 찾아서 실패 위치 전에 해당되는 TLB 엔트리를 선인출하는 선인출 명령어를 삽입하는 것이다. 이 방법은 프로그램에 대한 컴파일 시의 정보를 이용하여 컴파일 단계중 어셈블리 코드 생성 단계에서 구현될 수 있다.

본 논문에서는 선인출 명령어를 ptlb 라고 정의한다. ptlb 명령어는 일반적인 명령어와 동일한 형식을 가지며, 오퍼랜드는 선인출 하고자 하는 가상 페이지 번호를 가진다. ptlb 명령어가 수행되면, TLB를 접근하여 적중이 되면 NOP(No Operation)을 수행하고, 실패 시는 TPU를 구동시켜 오퍼랜드 값에 있는 가상 페이지 번호에 대한 번역값을 TLB에 저장하게 한다. TPU의 자세한 구현 방법은 다음 장에서 기술한다.

TLB 선인출을 수행하는 알고리즘은 알고리즘 1 및 2에서와 같다. 이 알고리즘은 다음의 두 가지 기본 원칙에 기반을 두고 있다. 첫째, 수식 (1), (2) 및 (3)에 의하여, 각 프로그램에서 I_1부터 I_n까지 순차 실패, 제어 실패, 데이터 실패를 초래하는 선인출 후보 들을 찾는다. 찾아진 선인출 후보들은 이전에 선택된 선인출 후보의 저장소인 AVL 트리에 존재 여부가 점검된다. 만약 AVL 트리에 존재하지 않으면, 순차 실패, 제어 실패, 데이터 실패를 초래하는 명령어 전에 ptlb 명령어를 추가하고, 차후의 비교를 위하여 AVL 트리에 저장한다. 만약 그렇지 않은 경우에는 ptlb 명령어를 추가하지 않는다. 일반적으로 찾아지는 선인출 후보들은 주소 공간에서 흩어져 있기 때문에, 배열(array) 구조가 색인 방법으로 사용된다면 많은 양의 기억장치 공간이 소비된다. 그래서, 본 연구에서는 기억장치의 공간적 요구사항과 합리적인 찾기 시간을 고려하여 AVL 트리를 사용하였다. AVL 트리의 평균 찾기 시간은 O(log n) 이다.

알고리즘 1.

```
Algorithm  insert_ptlb (virtual_page_number)
begin
sum = 0
forall instruction  Ii  in an object code do
        begin
        sum = instruction_length of  Ii  + sum
        if (sum is over page boundary)    /* check sequential miss */
        then
                check_avl (virtual_page_number of instruction address of Ii )
                sum = 0
        if (Ii  is branch instruction )          /* check control miss */
        then
                check_avl (virtual_page_number of branch address of Ii )
        if (Ii  is control instruction)
        then
                check_avl(virtual_page_number of implicit service address
                        of Ii )
        forall operand O1 , O2 , ... in Ii     /* check data miss.*/
                begin
                check_avl (virtual_page_number of operand of Ii )
                end
        end
end
```

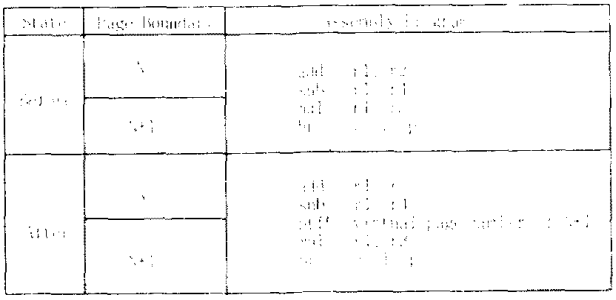
알고리즘 2.

```
Algorithm  check_avl (virtual_page_number)

begin
        if (virtual_page_number exist in AVL tree)
        then
                return
        else
                if (Ii is branch instructon or control instruction)
                then
                        insert ptlb instruction before
                        branch address of Ii
                else
```

```
                    insert ptlb instruction before ! /
        /* insert the virtual page number to AVL tree for next comparisons */
          insert_avl(virtual_page_number)
end
```

3.2 선인출 후보 찾는 방법

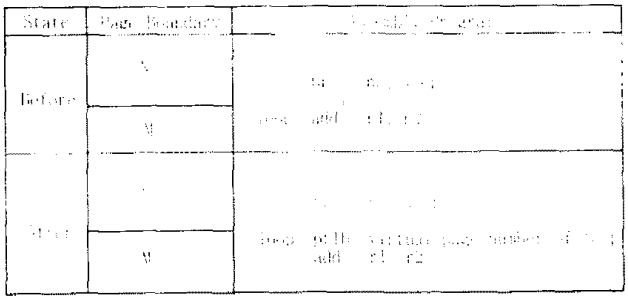
컴파일하는 동안에, 컴파일러는 각 명령어의 연산 코드(operation code) 부분을 해석하면 그 명령어의 길이를 알 수 있다. 프로그램의 시작점 부터 명령어의 길이를 누적하면 어느 명령어가 다음 페이지의 첫 번째 명령어인지를 알 수 있다. 따라서, 현재의 페이지 경계를 벗어나는 첫 번째 명령어 전에 ptlb 명령어를 삽입한다면 순차 실패를 방지할 수 있다. (그림 2)는 순차 실패의 방지를 위한 ptlb 명령어 삽입 예를 나타내고 있다. (그림 2)에서 mul 명령어는 새로운 페이지의 첫 번째 명령어이므로, TLB 접근 시 TLB 실패가 발생한다. 따라서, (그림 2)의 수정 상태와 같이 ptlb 명령어를 mul 명령어 전에 삽입하여 N +1 페이지에 대한 번역값을 선인출하여 TLB에 저장하게 하면 TLB 실패는 예방할 수 있다.



(그림 2) 순차실패 방지를 위한 ptlb 명령어 삽입 예
(Fig. 2) The example of ptlb instruction for sequential miss

제어 실패는 분기 및 제어 명령어의 특성에 의해 쉽게 파악된다. 분기 명령어의 제어는 실행 시에 정의된 조건의 값에 따라 분기 주소로 이전된다. 또, call, ret 명령어 등과 같은 제어 명령어의 서비스 주소는 컴파일 시의 정보에 의해 컴파일러가 파악 가능하며, 일부 제어 명령어는 컴퓨터 구조에서 정의된 암시적인 서비스 주소를 가지고 있다. 이 사실들은 바와 현재의 페이지 경계를 벗어나는 분기 명령어의 분기 주소나 제어 명령어의 암시적 서비스 주소에 있는 명령어가 수행되기 전에 ptlb 명령어를 추가한다면, TLB의 초기접근실패는 피할 수 있게 된다. 이 방법은 암시적인 서비스 주소를 갖는 제어 명령어와 조건을 갖지 않는 분기 명령어의 경우에는 제어 실패의 제거가 가능하나, 분기 조건을 갖는 명령어의 경우에는 목적 프로그램의 수행 과정에서 분기가 발생될 경우에만 제어 실패 제거가 가능하게 된다. 분기 조건을 갖는 제어 실패에 대한 ptlb 명령어의 추가 보기는 (그림 3) 및 (그림 4)와 같다.

(그림 3)과 같이 분기 주소가 새로운 페이지 경계의 최초에 있는 경우는 분기 명령어의 분기가 성공하는 경우와 실패하는 경우 모두 해결될 수 있다. 왜냐하면, ptlb 명령어는 분기 주소 부분에 위치하므로 분기가 발생할 경우는 제어 실패의 제거가 가능하며, 분기가 실패하더라도 목적 프로그램의 순차적 수행에 의하여 순차 실패를 제거할 수 있기 때문이다.

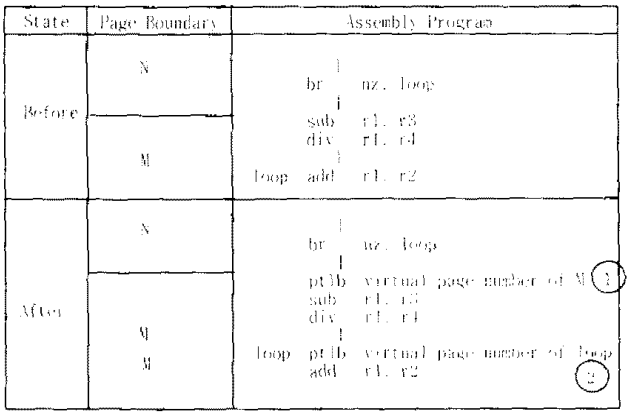


(그림 3) 분기 조건을 갖는 분기명령어의 경우의 ptlb 명령어 삽입 예 1
(Fig. 3) The example 1 of ptlb instruction for control miss with branch condition

그러나, (그림 4)와 같이 분기 주소가 새로운 페이지의 최초가 아닌 위치에 있을 경우는 분기가 실패할

경우에 제어 실패를 제거할 수 없게 된다. 목적 프로그램의 수행 과정에서 loop로 분기가 발생할 경우는 2 번 ptlb 명령어에 의하여 제어 실패가 제거될 수 있지만, 실패할 경우 2 번 ptlb 명령어는 NOP을 수행하게 된다. 왜냐하면, 페이지 M을 최초로 접근 시에 TLB 실패가 발생하여 기존의 TLB 실패 서비스 루틴에 의하여 페이지 M에 대한 번역값이 이미 TLB에 저장되어 있기 때문이다. 따라서, 이 경우 알고리즘 1과 2에 의해서는 ptlb 명령어가 추가되지 않지만, (그림 4)에서 1번 ptlb와 같이 순차 실패 발생 위치에 추가적으로 ptlb 명령어를 삽입하면 이 문제의 해결이 가능하다.

또, 제어 실패의 경우에서 분기 명령어의 loop가 다른 loop 안에 중첩되어 있고, 이 것이 페이지 경계 사이에 걸쳐 있는 특이한 경우에는 순차 실패, 제어 실패, 데이터 실패를 예방하기 위하여 추가한 ptlb 명령어가 loop 을 벗어나기 전까지는 불필요한 ptlb 명령어가 계속 수행되는 오버헤드를 초래할 수 있다. 이 경우, 최초 수행인 경우를 제외한 반복적인 ptlb 의 수행은 TLB 접근 시에 적중이 되기 때문에 NOP을 수행하게 된다. 따라서, ptlb 의 오버헤드는 수행될 때마다 한 개의 명령어 수행 시간 만큼 지연 효과를 초래한다. 따라서, 이와 같은 특이한 조건에서의 선인출 효과는 감소할 것으로 추측된다.



(그림 4) 분기 조건을 갖는 분기명령어의 경우의 ptlb 명령어 삽입 예 2

(Fig. 4) The example 2 of ptlb instruction for control miss with branch condition

데이터 실패 의 경우도 AVL 트리를 조사하는 방법으로 해결될 수 있다. 명령어의 오퍼랜드는 각 명령어가 점검될 때 마다 조사된다. 따라서, (알고리즘 1)과 (알고리즘 2)는 암시적 주소를 내포하지 않는 일부 제어 명령어를 제외한 대부분의 초기접근실패의 원인을 감소할 수 있으며, 그 결과 TLB의 초기접근실패가 감소된 만큼, 실패율은 감소하게 된다.

3.3 선인출 거리 적용 방법

지금까지는 목적 프로그램의 컴파일 단계에서 순차 실패, 제어 실패, 데이터 실패를 초래하는 선인출 후보를 찾아서, TLB 실패 서비스를 위한 ptlb 명령어를 추가하는 방법에 대해 기술하였다. 이 방법은 TLB 실패 서비스를 ptlb 명령어와 TPU 라고 하는 H/W을 사용하여 S/W 관리 TLB의 실패 서비스 시에 초래되는 문맥 교환 시간과 주기억장치로 부터의 TLB 실패 서비스 시에 필요한 명령어 인출 시간을 감소시키는 것이다. 그러나, 제안하는 선인출 기법은 ptlb 명령어를 삽입하는 위치에 따라 추가적인 성능 개선을 얻을 수 있다.

TLB 실패 서비스 프로그램을 수행하는데 소요되는 시간을 선인출 거리(prefetch distance)라고 하자. 이 것은 TLB 엔트리의 선인출을 위한 TLB 서비스 프로그램의 수행이 적어도 TLB 실패 서비스 프로그램을 구성하는 명령어 수 이전에 수행되어야 함을 의미한다. 만약, 순차 실패, 제어 실패, 데이터 실패를 초래하는 선인출 후보 명령어로 부터 선인출 거리 사이에 다른 곳으로 분기하거나 또는 선인출 거리 밖의 명령어로 부터 선인출 거리 내로 분기되어 오는 명령어가 없는 경우에, ptlb 명령어를 선인출 거리 전에 삽입을 한다면 TLB 실패 서비스로 인한 목적 프로그램의 정지(stall) 시간을 제거할 수 있다. 이 방법은 컴파일 시에 얻는 목적 프로그램에 대한 지식을 이용하여 처리될 수 있다.

(알고리즘 3)은 선인출 거리를 고려하는 경우에 (알고리즘 2)를 대체하는 알고리즘이다. 선인출 거리의 적용은 분기 조건을 갖는 분기 명령어 경우를 제외한 모든 순차 실패, 제어 실패, 데이터 실패를 초래하는 선인출 후보 명령어에 적용이 가능하다. 또, 선인출 거리 내에 분기 조건을 가지는 경우는 선인출 후보 명령어로 부터 역 추적(backtracking)하여 분기 조건이 없는 가장 먼 위치에 ptlb 명령어를 추가한다면,

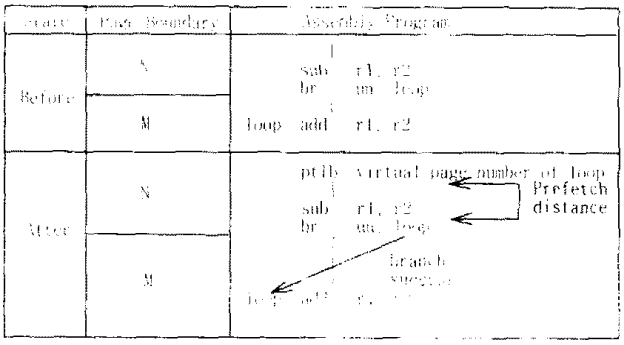
```
알고리즘 3.
Algorithm  check_avl (virtual_page_number)

begin
        if (virtual_page_number exist in AVL tree)
        then
                return
        else
                if (branch conditions not exist
                      between prefetch distance from Ii AND
                      Ii is not conditional branch instruction)
                then
                        insert ptlb instruction before
                        prefetch distance from Ii
                else
                        /* check conditional branch instruction */
                        if (Ii is conditional branch instructon)
                        then
                                insert ptlb instruction before
                                   branch address of Ii
                        else
                                check first Ii-n that do not have
                                   branch condition from Ii
                                   by backtracking,
                                   then, insert ptlb instruction
                                   before Ii-n
       /* insert the virtual page number to AVL tree for next comparisons */
        insert_avl(virtual_page_number)
        return
end
```

선인출 후보 명령어로 부터 미리 수행되는 만큼의 TLB 실패 서비스 시간을 단축할 수 있다.

(그림 5)는 분기 조건이 없는 분기 명령어의 경우의 ptlb 명령어 삽입 예를 나타내고 있다. (그림 5)의 예는 다른 페이지에 존재하는 loop로 무조건 분기하기 때문에 선인출 거리 내에 다른 곳으로 분기 명령어가 없다면, 선인출 거리 전에 ptlb 명령어를 수행하는 것은 타당함을 알 수 있다. 또, 순차 실패 및 데이터 실패의 경우도 선인출 거리 내에 다른 곳을 분기할 경우가 없으면 선인출 거리 전에 ptlb 명령어를 삽입하는 것은 같은 개념으로 적용이 가능하다.

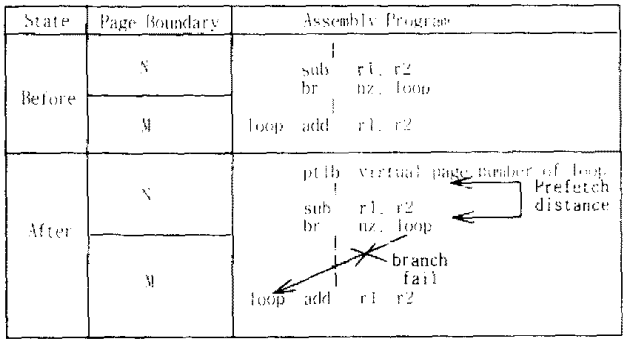
그러나, (그림 6)과 같이 분기 조건을 갖는 분기 명령어의 경우는 목적 프로그램의 수행 과정에서 분기 조건의 값에 따라 선인출 거리를 고려한 것이 유효할

수도 있으나, 분기가 발생하지 않을 경우는 TLB 실패를 초래하지 않음에도 불구하고 분기주소에 대한 번역값을 TLB에 저장하게 되므로, 이 경우는 선인출 거리를 반영하는데 문제가 발생할 수 있다. 그러나, 이 경우도 ptlb 명령어가 수행되면 LRU 알고리즘에 의하여 가장 최근에 사용되지 않은 엔트리를 교체하기 때문에 TLB 엔트리 수가 많은 경우에는 동일 프로그램의 수행 환경에서 가상기억장치가 갖는 구역성(locality)과 작업 세트(working set) 개념을 고려하면 유효할 수도 있을 것으로 추측되나, 실제적인 성능 개선 유무는 S/W 제어 선인출 기법을 실제적으로 구현하고, 그 결과를 분석하여야 판단할 수 있을 것이다.



(그림 5) 분기 조건이 없는 분기 명령어 경우의 ptlb 명령어 삽입 예

(Fig. 5) The example of ptlb instruction for control miss without branch condition



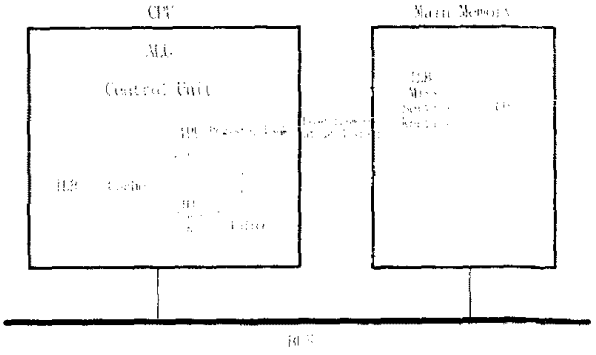
(그림 6) 분기 조건이 있는 분기 명령어에서 선인출거리 고려 시 문제 발생 보기

(Fig. 6) The example of ptlb instruction for control miss without branch condition

본 절에서 제시된 선인출 후보 및 선인출 거리 개념을 사용할 경우에 TLB 실패의 종합적 처리 방법 및 효과는 다음과 같다. 제안된 알고리즘에 의해 찾아진 선인출 후보에 대해 선인출 거리 전에 ptlb 명령어를 삽입할 수 있는 경우는 TPU의 선인출에 의하여 TLB 실패 서비스 자체를 제거할 수 있고, 선인출 거리 내에 ptlb 명령어를 삽입할 수 있는 경우는 문맥 교환 및 복구 시간과 선인출 후보로 부터 ptlb 명령어가 미리 수행되는 만큼 TLB 실패 서비스 시간은 감소시킬 수 있다. 또, 선인출 거리 개념 적용이 불가능한 경우는 선인출 후보 전에 ptlb를 추가함으로써 TLB 실패 시 필요한 문맥 교환 시간 및 복구 시간과 명령어 인출 시간의 감소 효과를 얻을 수 있다. 나머지 경우, 즉, ptlb 명령어가 추가되지 않은 초기접근실패, 용량접근실패, 충돌접근실패에 대해서는 트랩에 의하여 기존 S/W 관리 TLB의 TLB 실패 서비스 루틴에 의하여 서비스된다.

4. 선인출 명령어 및 TPU의 구현

본 장에서는 3 장에서 제안한 선인출 명령어 및 TPU가 CPU 내에서 구현될 수 있는 방법을 제시하다. 제안된 ptlb 명령어의 수행은 프로세서 내의 특별한 TPU H/W나 또는 독립된 협동프로세서(coprocessor)에 의해 처리될 수 있다. ptlb 명령어는 CISC 구조에서 지원되는 명령어 수행에 의하여 TLB를 직접 갱신하는 TLB 갱신 명령어와는 동작 방법이 다르다. ptlb 명령어는 오퍼랜드로 주어진 가상 페이지 번호로



(그림 7) TPU의 개념도

(Fig. 7) The conceptual diagram of TPU

TLB를 접근하여 적중이 되면 NOP을 수행하고, 실패 시는 단지 TPU H/W를 동작시키는 역할을 수행한다. TPU의 개념도는 (그림 7)과 같다. TPU의 기능은 선인출 거리를 고려하는 경우와 고려하지 않는 경우에 차이가 있다.

4.1 선인출 거리를 고려하지 않는 경우

TPU는 레지스터 뱅크와 한 개의 카운터로 구성이 된다. 레지스터 뱅크는 완전한 TLB 실패 서비스 루틴의 저장과 TLB 실패 서비스 루틴의 수행 시 발생되는 명령어의 중간 결과를 저장하는데 사용된다. 카운터는 TLB 실패 서비스 루틴에서 다음 수행할 명령어를 가리키는 명령어 포인터로 사용된다. 시스템의 초기 시동 시에 완전한 TLB 실패 서비스 루틴이 운영체제에 의해 TPU로 저장되며, 카운터는 TLB 서비스 루틴의 명령어 수 만큼 미리 저장된다. 이 때, 일반적으로 S/W 관리 TLB의 서비스 루틴이 포함하는 문맥 교환을 위한 프로그램의 상태 보존과 서비스 수행 후의 상태 복귀에 필요한 코드는 제외된다. 일반적으로 TLB 실패 서비스 루틴은 어셈블리 코드로 작성이 되므로, 프로세서 내부에서 레지스터 자원의 사용은 제한될 수 있다. 따라서, 본 연구에서는 TLB 실패 서비스 루틴의 각 명령어는 오직 TPU 내의 레지스터 뱅크만을 접근한다고 가정한다. 이와 같은 가정에 의해, TLB 선인출 과정 동안 문맥 교환의 필요성은 제거될 수 있다.

시스템이 동작할 때, 카운터는 레지스터 뱅크에 저장되어 있는 TLB 실패 서비스 루틴의 시작점을 가리키고 있다. 정상적인 상태 (즉, TLB 접근이 성공할 때)에서 프로세서 내부의 제어 장치(control unit)는 명령어를 인출하기 위하여 캐시를 접근하지만, 만약 ptlb 명령어가 발생되면 캐시 장치 대신에 TPU의 레지스터 뱅크로 부터 명령어를 인출한다. TPU로 부터 명령어 인출이 될 때마다 카운터는 1씩 감소하여, 0이 되면 제어는 정상 상태로 넘어가게 된다. 이 때 카운터는 다음에 발생할 TLB 실패 처리를 위하여 TLB 실패 루틴의 처음을 가리키게 된다.

또, 다른 ptlb 명령어의 요구사항은 ptlb 명령어는 프로세서 내부의 사용자 상태 또는 커널 상태 등에서 예외 없이 수행될 수 있어야 하는 것이다. 다시 말하면, H/W 관리 TLB 에서 처럼 문맥 교환이 일어나지 않아야 한다. 이와 같이 구현함으로써, 시스템 내부의 다른 자원들은 TLB 선인출 동작과 무관함이 보장될 수 있다.

4.2 선인출 거리를 고려하는 경우

선인출 거리를 고려하는 경우에도 기본적인 TPU의 개념은 동일하다. 선인출 거리를 고려하지 않는 경우에 TPU는 TLB 실패 서비스 루틴을 저장하고, ptlb 명령어가 수행되면 레지스터 뱅크에 저장하고 있는 TLB 실패 서비스 루틴의 각 명령어를 CPU의 제어 장치로 공급하는 역할만 수행한다. 그러나, 선인출 거리를 고려하는 경우는 ptlb 명령어에 의해 TPU가 구동되면 TPU 내부에 저장하고 있는 TLB 실패 서비스 루틴의 명령어를 TPU 내에서 자체적으로 수행하는 능력을 가져야 한다는 점이다. 따라서, TPU는 구동이 시작되면 CPU와 병렬로 수행되는 기능을 가지게 된다.

또, 선인출 거리 내에 분기 조건을 가져서 ptlb 명령어가 선인출 거리 내에서 수행이 되는 경우를 위하여, CPU의 제어장치와 TPU 간에 H/W로 구현되어야 하는 interlocking 기법이 필요하다. TLB 실패 발생 시에 TPU가 동작 중이면 CPU는 운영체제에게 TLB 실패 서비스 요청을 하지 않고 상태 플랙에 대기 중이라는 플랙을 표시하고 잠시 수행을 중단(stall)하는 기능이 필요하며, 이 경우에 TPU는 상태 플랙의 점검을 통하여 CPU의 계속적 수행이 가능하도록 알려주는 기능이 필요하다. 다른 방법으로는 RISC의 경우에 TPU의 동작 종료 시간을 계산하여 필요한 만큼 컴파일 단계에서 NOP을 추가함으로서도 해결될 수도 있다.

이와 같은 기능을 지원함으로써, 선인출 거리 내에서 ptlb 명령어가 수행되어도 TPU는 현재 상황에 적절한 동작을 할 수 있게 된다.

5. 정량적 분석

이 절에서는 선인출하는 TLB 엔트리가 있을 경우 본 연구에서 제안하는 기법이 선인출을 하지 않는 경우보다 전체 수행 시간이 감소함을 정량적으로 분석한다.

목적프로그램에 대한 전체 수행 시간 비교를 간략

화 하기 위하여, 캐시 접근 실패와 외부 인터럽트에 의한 문맥 교환은 선인출 기법과는 무관하게 발생되므로 고려 대상에서 배제한다. 목적프로그램의 수행 시간을 비교하기 위하여 다음과 같이 정의한다.

1. ET(Execution Time):정상적인 경우의 목적프로그램을 수행할 때 소요되는 전체 수행 시간을 말한다.
2. PET(Prefetch Execution Time):S/W 제어 선인출 기법을 적용하였을 경우의 목적프로그램을 수행할 때 소요되는 전체 수행시간을 의미한다.
3. NI(Number of Instruction):목적프로그램을 구성하는 명령어 수를 말한다.
4. AIET(Average Instruction Execution Time):목적프로그램을 구성하는 명령어의 평균 수행 시간을 말한다.
5. TMR(TLB Miss Rate):목적프로그램 수행 과정에서 발생하는 TLB의 실패율을 말한다.
6. TMP(TLB Miss Penalty):TLB 실패 서비스에 소요되는 시간을 말한다.
7. PTMR(Prefetch TLB Miss Rate):S/W 제어 선인출 기법을 적용하였을 때의 목적프로그램 수행 과정에서 발생하는 TLB의 실패율을 말한다.
8. NPI(Number of ptlb instruction):목적프로그램에 추가되는 ptlb 명령어의 수를 말한다.
9. PIET(Ptlb Instruction Execution Time):ptlb 명령어 수행으로 TLB에 접근하여 실패가 되었을 때, TPU를 구동시켜 입력으로 받은 가상페이지 번호에 대한 번역값을 TLB에 저장하는데 소요되는 시간을 의미한다.
10. PNOP(Ptlb No Operation):ptlb 명령어 수행으로 TLB 접근 시 적중이 되어 NOP을 수행하는데 소요되는 시간을 의미한다.
11. NNOP(Number of NOP):NOP이 수행되는 회수를 의미한다.

임의의 목적프로그램에 대한 전체 수행 시간 ET는 명령어 수행시간과 TLB 실패 서비스 시간의 합으로 수식 (5)와 같이 표현할 수 있다.

$$ET = NI*AIET + NI*TMR*TMP \quad (5)$$

그리고 선인출 기법을 사용하는 경우의 전체 수행 시간인 PET는 명령어 수행시간(AIET), ptlb 명령어 수행시간(PIET), ptlb 명령어 수행으로 감소된 TLB 실패율(PTMR)에 의한 TLB 실패 서비스 시간(TMP), 그리고 분기 예측 실패로 추가적으로 수행되는 ptlb 명령어 수행시간 (PNOP*NNOP)의 합으로 수식 (6)과 같이 표현될 수 있다.

$$PET = NI*AIET + (NPI - NNOP)*PIET + NI*PTMR*TMP + PNOP*NNOP \quad (6)$$

S/W 관리 TLB에서 선인출 기법에서 발생한 TLB 실패 회수를 n이라고 하고, 선인출 기법을 사용하지 않은 경우의 TLB 실패 회수를 m이라고 하면, 선인출을 하지 않은 경우의 TLB 실패율은 TLB 실패 빈도수를 목적 프로그램의 명령어 수로 나눈 결과로 식 (7)과 같으며, 선인출을 하는 경우의 TLB 실패율은 같은 방법으로 표현하면 식 (8)과 같다.

$$TMR = m/NI \quad (7)$$

$$PTMR = n/NI \quad (8)$$

선인출 기법을 적용하는 경우 TLB 실패가 감소하는 만큼 ptlb 명령어가 추가되며 선인출 예측이 실패하였을 NOP을 수행하는 회수를 삭감하면 되므로 NPI 는 (m−n+NNOP)이다. 수식 (5)에서 수식 (6)의 차이를 구하면,

$$ET - PET = NI*TMP*(TMR - PTMR) - (NPI - NNOP)*PIET - PNOP*NNOP \quad (9)$$

와 같다. 만약 정상적인 경우와 선인출 기법을 적용하였을 때의 차인 수식 (9)가 양수이면, 선인출 기법의 성능 개선 효과가 있는 것이다. 다음 단계로 TMR, PTMR 및 NPI를 수식 (7), (8), 그리고 (m−n−NNOP)으로 대치하면

$$ET - PET = (m-n)*(TMP - PIET) - NNOP*PNOP \quad (10)$$

을 얻는다. 여기서 n은 순차 실패, 제어 실패, 데이터 실패의 경우에 발생되는 TLB 실패를 감소한 것이고, NNOP은 제어 실패에만 관계되므로 $n > NNOP$이다.

또, $m > n$ 이다. 따라서,

$$(m-n) > NNOP \quad (11)$$

이다. 그리고, S/W 관리 TLB에서 TMP는 PIET보다 TLB 실패 서비스 루틴 수행 시 추가적인 문맥 교환과 서비스 후의 이전 문맥 복구 시간이 추가로 소비되고, TLB 실패 서비스 루틴이 명령어의 집합으로 구성되므로 명령어 인출이 추가적으로 더 소요되므로 더 크다. 또, 선인출 거리를 고려하여 PIET가 TPU 동작에 소요되는 시간 전에 수행이 된다면 PIET는 정상적인 명령어 수행과 병렬로 수행되기 때문에 명령어 한 개 수행 시간과 같게 된다.

$$TMP - PIET = TMP - 1 \approx TMP \quad (12)$$

또, TMP는 문맥 교환 및 실패 서비스가 다수의 명령어에 의해 처리되고, PNOP은 NOP을 수행하는 한 개의 단순한 명령어 수행 시간과 같으므로,

$$TMP \gg PNOP = 1 \quad (13)$$

이다. 따라서, 식 (11)과 (13)에 의해서 식 (10)은 양수이며, 이에 따라 선인출 기법이 목적 프로그램의 전체 수행 시간을 감소시키는 유효한 기법임을 알 수 있다.

또, 성능 개선 정도는 식 (10)에서 유추할 수 있다. 예를 들어, DECstation 3100 에서 RISC인 R2000/R3000의 문맥 교환 오버헤드가 103 명령어[12]인 점을 고려한다면, TMP는 PIET보다 103개 명령어 만큼 많으므로 ptlb 명령어의 수행 과정에서 TLB 적중이 되어 NOP을 수행하는 경우가 어느 정도 발생한다고 할 지라도 전체적인 관점에서 보면 성능 개선 효과 정도를 추측할 수 있다. 또한, 선인출 거리를 고려하는 경우에 PIET를 감소할 수 있으므로 추가적인 개선 효과가 있는 것을 알 수 있다.

6. 결 론

S/W 관리 TLB는 H/W 설계를 단순화시킬 뿐만 아니라, 페이지표의 구조에 상당한 융통성을 제공하여 개방형 환경에서 임의의 프로세서에서 요구되는 다양한 운영체제의 이식을 용이하게 한다.

본 논문에서는 S/W 관리 TLB의 초기접근실패를 감소하기 위한 S/W 제어 선인출 기법을 제안하고, 제안된 기법의 이론 및 구현방법을 제시하였고, 정량적 분석에 의해 제안된 기법이 S/W 관리 TLB 의 성능 향상에 유효한 기법임을 보였다. 제안하는 프트웨어 제어 선인출 기법은 S/W 제어 선인출 기법은 TLB의 초기접근실패를 감소시키는 방법으로는 처음 제안된 것으로, TLB 실패율은 감소시키나, TLB 접근시간 증가를 초래하지 않는 유효한 기법으로 판단된다. 또, 이 기법은 S/W 관리 TLB의 융통성을 그대로 제공하면서, 특정한 응용프로그램에만 적용될 수 있는 것이 아니라, 운영체제를 포함한 모든 종류의 프로그램에서 TLB 실패 패널티를 감소시키는데 적용가능하고, 추가적으로 버스 경쟁도 감소시키므로써 대칭형 다중 처리 시스템, 병렬 처리 시스템 등 버스 경쟁의 완화가 중요한 문제가 될 수 있는 분야의 연구에 상당한 개선 효과를 줄 것으로 판단된다.

본 논문에서 제안하는 S/W 제어 선인출 기법은 TLB 파라미터 변경에 의한 기존의 성능 향상 방법과는 해결 목표가 서로 상이하다. 따라서, 앞에서 언급한 H/W 기반의 여러 기법과 혼용될 수 있으며, 혼용시는 추가적인 S/W 관리 TLB의 성능 개선을 기대할 수 있다. 또, 선인출 거리 개념과 TPU를 이용하여 TLB 실패를 발생시키는 명령어 이전에 선인출 명령어를 삽입하여, 필요한 TLB 번역값을 미리 TLB에 저장하게 하는 기본 개념은 H/W 관리 TLB에서 초기접근실패의 성능 향상 방법으로도 활용될 수 있다.

향후 과제로는 목적 프로그램에 S/W 선인출 기법의 실제적 구현과 성능 평가 작업이 필요할 것으로 판단된다. 또, 현재 일부 프로세서에서 지원되고 있는 TLB에서의 다중 페이지 지원 환경에서 S/W 제어 선인출 기법의 지원 방법 등에 대한 연구도 필요할 것으로 전망된다.

참 고 문 헌

[1] D.W. Clark and J.S. Emer, Performance of the VAX-11/780 Translation Buffer:Simulation and Measurement, ACM Transactions on Computer

Systems, vol. 3, no. 1, pp. 31-62, 1985.

[2] J.B. Chen, A. Borg, and N.P. Jouppi, A simulation based study of TLB performance, The 19th Annual International Symposium on Computer Architecture, 1992.

[3] M. Talluri, S. Kong, M. D. Hill, and D. A. Patterson, Tradeoffs in supporting two page sizes, The 19th Annual International Symposium on Computer Architecture, 1992.

[4] D. Nagle, R. Uhlig and T. Stanley, Design Tradeoffs for Software-Managed TLBs, The 20th Annual International Symposium on Computer Architecture, 1993.

[5] R. Uhlig, D. Nagle, T. Stanley, T. Mudge, S. Sechrest, and R. Brown, Design Tradeoffs for Software-Managed TLBs, ACM Transactions on Computer Systems, vol. 12, no. 3, pp. 175-205, 1994.

[6] M. D. Hill and A. J. Smith, Evaluating Associativity in CPU Caches, IEEE Transactions on Computer, vol. 38, pp. 1612-1630, 1989.

[7] C. A. Alexander, W. M. Keshlear, and F. Briggs, Translation buffer performance in a UNIX environment, Computer Architecture News, vol. 13, no. 5, pp. 2-14, 1985.

[8] W. Stallings, Operating Systems, Macmillan Publishing Company, 1992.

[9] D. A. Patterson and J. L. Hennessy, Computer Organization & Design: The Hardware/Software Interface, Morgan Kaufmann Publishers, Inc., 1994.

[10] J. S. Park and G. S. Ahn, A Software-Controlled Prefetching Mechanism for Software-Managed TLBs, Microprocessing and Microprogramming Journal, vol. 41, no. 2, pp. 121-136, 1995.

[11] D. A. Patterson and J. L. Hennessy, Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, Inc., 1990.

[12] T. E. Anderson, H. M. Levy, B. N. Bershad, and E. D. Lazowska, The interaction of architecture and operating system design, The Fourth International Conference on Architecture Support for Programming and Operating Systems, 1991.

박 장 석

1982년 경북대학교 전자공학과 (학사)

1985년 경북대학교 전자공학과 (석사)

1995년 경북대학교 컴퓨터공학과(박사)

1983년~1995년 한국전자통신연구원 컴퓨터연구단 선임연구원

1990년 정보처리기술사

1995년~현재 정보통신연구관리단 정보기술평가 1실장(책임연구원)

관심분야: 컴퓨터 구조, 멀티미디어 시스템, 소프트웨어 공학