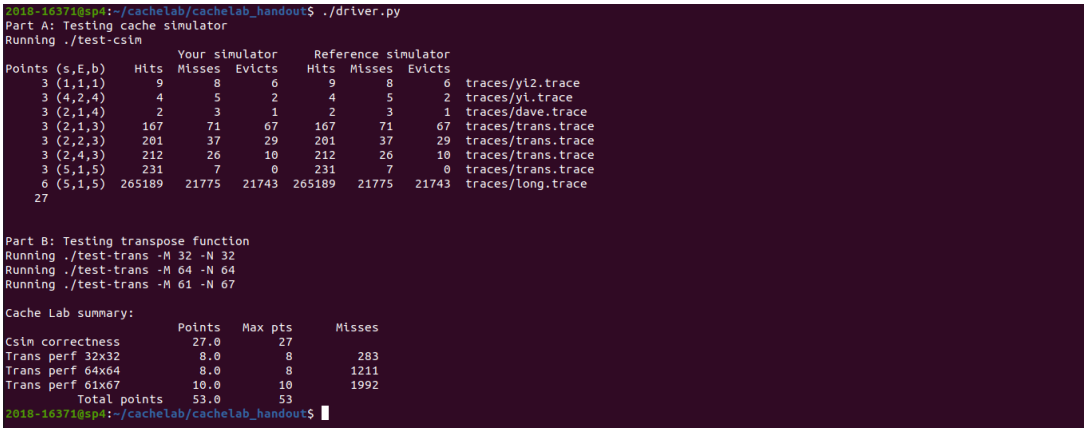
시스템 프로그래밍 Lab 4

2018-16371
자유전공학부 문보설

1. 실행 결과



2. 구현 방법

A. 캐시 구현

cache는 캐시의 한 라인을 나타내는 structure를 만들어서 해당 structure의 배열로 구현하였다. cache block number나 set number는 각각 캐시의 열, 행에 대응된다. 그리고 캐시의 한 라인을 나타내는 structure에 tag, valid, lru counter를 관리하여 캐시에 저장된 블락 정보를 알 수 있게 하였다. 기본적인 load, store, modify 알고리즘에 따라 구현하였으며, eviction에는 lru counter를 활용했다. lru counter는 한번의 캐시 접근에 해당 캐시라인을 접근하지 않으면 1커진다. 접근된 캐시라인은 lru 카운터가 0이 된다. 즉, lru counter는 최근 캐시라인을 접근하지 않은 연속적인 연산들의 개수라고 할 수 있다. 따라서 이 값이 클수록 가장 오래 전에 접근한 캐시라인이라고 보고, 해당 캐시라인을 eviction대상으로 삼았다.

B. Matrix Transformation 구현

이 부분은 block으로 transformation하는 알고리즘을 따르되 32*32와 64*64, 61*67 각각의 경우를 나누어 구현했다. 32*32의 경우와 64*64의 경우, block size를 8로 하되 최대한 cache가 되어 있는 부분을 활용하기 위해 추가적인 구현 방식을 사용했다. 하나의 block이 4번째 행과 5번째 행 사이, 4번째 열과 5번째 열 사이를 기준으로 4분면으로 나뉜다고 해 보자. 그러면 먼저 A의 block 1, 2사분면을 접근하면서, A의 1사분면은 B의 1사분면 적절한 곳에 매핑하고 2사분면은 B의 2사분면에 임시로 저장했다. 다음으로 A의 3, 4분면을 접근하면서 앞서 임시로 활용한 B의 2사분면을 B의 3사분면 적절한 곳에 매핑하고, B의 2사분면은 A의 3사분면으로 채운다. 동시에 A의 4사분면은 B의 4사분면에 매핑한다. 61*67의 경우 block size를 16으로 하니 조건을 만족하는 cache miss가 나서 16 sized

block을 활용해 transform했다.

3. 어려웠던 점

   Matrix Transformation의 cache miss를 줄이는 부분이 가장 어려웠다. 먼저 최적화 방법을 생각해 내는 것이 어려웠다. 특히 32*32, 64*64 matrix transformation에서는 block size를 조정하는 것만으로 충분히 cache miss를 줄일 수 없어서 어려웠다. 캐시가 어떻게 동작하는지를 잘 생각해야 최적화 방법을 생각해 낼 수 있었다. 그리고 과제 stack spec에 맞는 최적화를 생각해 내는 것도 어려운 부분 중 하나였다. 내가 생각한 최적화는 한번 캐시된 8개의 원소를 다 올바르게 매핑하거나 다른 곳에 저장했다가 나중에 이 저장내용만을 바탕으로 적절한 곳에 매핑할 수 있도록 하는 것이었기 때문에, 많은 local variable이 필요했다. 따라서 이번 과제의 stack spec에 맞추기 위해 이전에 정의한 variable들을 적절하게 활용하는 것이 중요했다. 이 두 가지에 가장 신경을 많이 썼고, 이 두 가지가 이번 과제를 하면서 가장 어려웠던 점인 것 같다.

4. 새롭게 배운 점

   캐시의 동작에 대해서 더 깊게 알 수 있었고, cache miss를 최소화하는 전략에 대해서 배울 수 있었다.