성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

[REPORT]

■과 목 명: 운영체제 (SWE3004-42)
■담 당 교 수: 신동군 교수님
■제 출 일: 2022년 3월 28일
■학 과: 수학과
■학 번: 2016314786
■성 명: 김호진

Chap 19. TLB Measurement

수학과 김호진 (2016314786)

1. For timing, you'll need to use a timer (e.g., gettimeofday()**). How precise is such a timer? How long does an operation have to take in order for you to time it precisely? (this will help determine how many times, in a loop, you'll have to repeat a page access in order to time it successfully)**

Timer로 gettimeofday()를 사용하는 경우, 최소 단위는 microsecond(1×10^{-6}초)이다.

만약 nanosecond(1×10^{-9}초)의 정밀도로 결과값을 얻고 싶다면, 측정을 반복하여 평균값을 산출해야 하기 때문에 1000(=10^3)번의 반복 횟수가 필요하다.

2. Write the program, called tlb.c**, that can roughly measure the cost of accessing each page. Inputs to the program should be: the number of pages to touch and the number of trials.**

다음과 같은 소스코드로 작성된 프로그램 tlb.c는 페이지 수와 시행 횟수를 입력인자로 받아 각 페이지에 대한 액세스 비용을 대략적으로 측정한다.

home > borussen > C tlb.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>

#define PAGESIZE 4096

int main(int argc, char *argv[])
{
    if (argc != 3) {
        fprintf(stderr, "USAGE: tlb <number of pages> <number of trials>\n");
        return 1;
    }

    int numPages = atoi(argv[1]);
    int numTrials = atoi(argv[2]);

    if (numPages <= 0 || numTrials <= 0) {
        fprintf(stderr, "Invalid input\n");
        return 1;
    }

    struct timeval start, end;
    int jump = PAGESIZE / sizeof(int);
    int *a = malloc(numPages * jump * sizeof(int));

    for(int i = 0; i < numPages * jump; i++) {
        a[i] = i;
    }

    gettimeofday(&start, NULL);
    for (int i = 0; i < numTrials; i++) {
        for (int j = 0; j < numPages * jump; j += jump) {
            a[j] += 1;
        }
    }
    gettimeofday(&end, NULL);

    // nanoseconds
    long time = (end.tv_sec - start.tv_sec) * 1000000000 + (end.tv_usec - start.tv_usec) * 1000;
    double average = time / (numPages * numTrials);
    printf("%f\n", average);

    free(a);

    return 0;
}
```

3. Now write a script in your favorite scripting language (bash?) to run this program, while varying the number of pages accessed from 1 up to a few thousand, perhaps incrementing by a factor of two per iteration. Run the script on different machines and gather some data. How many trials are needed to get reliable measurements?

앞서 작성한 프로그램 tlb.c를 실행하기 위해 대표적인 scripting language인 python을 활용하였다. 액세스하는 페이지 수와 실행 횟수를 입력인자로 받았고 문제의 요구사항에 따라 페이지 수를 1(=2^0)부터 8192(=2^{13})까지 2배씩 증가하여 측정하였다. 신뢰할 수 있는 결과를 얻기 위해서 실행 횟수는 100,000번으로 잡았다.

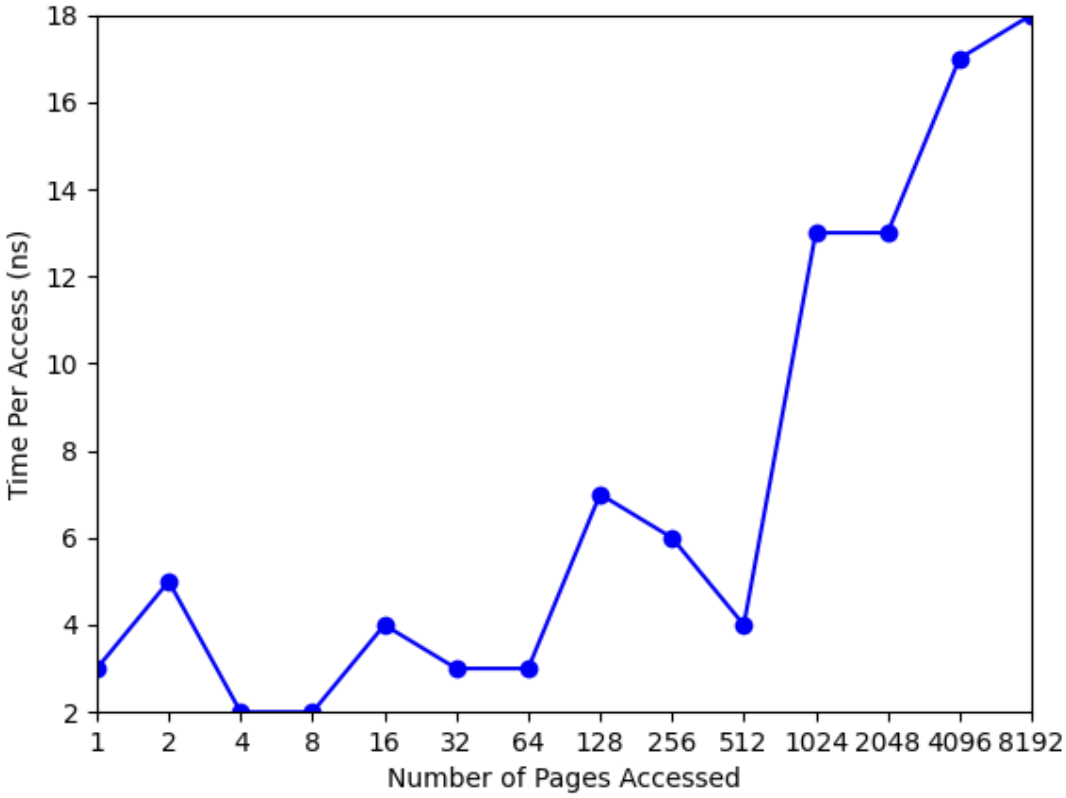
home > borussen > plot.py

```python
import argparse
import subprocess
import matplotlib.pyplot as plt
import numpy as np

parser = argparse.ArgumentParser()
parser.add_argument('pages', type=int)
parser.add_argument('trials')
args = parser.parse_args()

data = []
x = np.arange(args.pages)
pages = 2 ** x

for i in pages:
    r = subprocess.run(['./tlb.out', str(i), args.trials],capture_output=True, check=True, text=True)
    data.append(float(r.stdout))

plt.margins(0)
plt.plot(x, data, 'bo-')
plt.xticks(x, pages)
plt.xlabel('Number of Pages Accessed')
plt.ylabel('Time Per Access (ns)')
plt.savefig('result.png')
plt.show()
```

```
borussen@DESKTOP-L834KLC:~$ make
cc -O0 -o tlb.out tlb.c -Wall
borussen@DESKTOP-L834KLC:~$ python3 plot.py 14 100000
```

4. Next, graph the results, making a graph that looks similar to the one above. Use a good tool like ploticus **or even** zplot**. Visualization usually makes the data much easier to digest; why do you think that is?**

실행결과를 그래프를 이용하여 시각화하기 위해서 python의 numpy 라이브러리와 Matplotlib 라이브러리를 활용하였고, 다음의 결과를 얻었다.



이처럼 그래프를 활용하면 데이터의 경향성을 쉽게 파악할 수 있고 결과를 직관적으로 받아들일 수 있기 때문에 시각화(Visualization)는 데이터 이해에 있어 유용하다.

5. One thing to watch out for is compiler optimization. Compilers do all sorts of clever things, including removing loops which increment values that no other part of the program subsequently uses. How can you ensure the compiler does not remove the main loop above from your TLB size estimator?

컴파일러가 TLB size estimator에서 main loop를 제거하지 않도록 하기 위해서는 컴파일러 최적화를 비활성화 해야 하고, 이를 위해서 gcc의 optimize 옵션인 gcc -O0를 사용한다.

6. Another thing to watch out for is the fact that most systems today ship with multiple CPUs, and each CPU, of course, has its own TLB hierarchy. To really get good measurements, you have to run your code on just one CPU, instead of letting the scheduler bounce it from one CPU to the next. How can you do that? (hint: look up "pinning a thread" on Google for some clues) What will happen if you don't do this, and the code moves from one CPU to the other?

Linux 상에서 스케줄러가 CPU를 다른 CPU로 bounce 하는 대신 하나의 CPU 상에서 코드를 고정적으로 실행하려면 프로세스는 <sched.h>에 있는 'sched_setaffinity()', 'sched_getaffinity()' function을, 쓰레드는 <pthread.h>에 있는 'pthread_setaffinity_np()'와 'pthread_getaffinity_np()' function을 각각 사용하면 된다.

만약 코드가 CPU 사이에서 이동을 하여 여러 CPU 상에서 실행되면, TLB의 성능을 제대로 측정하기 어렵다.

7. Another issue that might arise relates to initialization. If you don't initialize the array a above before accessing it, the first time you access it will be very expensive, due to initial access costs such as demand zeroing. Will this affect your code and its timing? What can you do to counterbalance these potential costs?

문제에서 언급되었듯이 배열에 접근하기 전 배열을 초기화하지 않으면, demand zeroing 등의 초기 액세스 비용이 들기 때문에 처음 액세스할 때 많은 비용이 들게 된다. 그러므로 배열이 사전에 올바르게 초기화되지 않으면 프로그램의 performance에 영향을 줄 가능성이 높다.

따라서 이러한 잠재적 비용을 상쇄시키고 정확한 측정을 하기 위해서는 시간을 측정하기 이전에 calloc() 등을 사용하여 배열을 초기화해주는 것이 좋다.