sum.c

```c
#include<stdio.h>
#include<stdlib.h>

#define COLS 10000
#define ROWS 10000

int sum_array_rows(int** a)
{
    int i, j, sum = 0;

    for (i = 0; i < ROWS; i++)
        for (j = 0; j < COLS; j++)
            sum += a[i][j];
    return sum;
}

int sum_array_cols(int** a)
{
    int i, j, sum = 0;

    for (j = 0; j < COLS; j++)
        for (i = 0; i < ROWS; i++)
            sum += a[i][j];
    return sum;
}

int main()
{
    int** a;
    int i, j;

    a = (int**)malloc(sizeof(int*) * ROWS);

    for(i = 0; i < ROWS; i++)
        a[i] = (int*)malloc(sizeof(int)*COLS);

    for(i = 0; i < ROWS; i++)
        for(j = 0; j < COLS; j++)
            a[i][j] = i;

#ifdef SUM_COLS
    int sum = sum_array_cols(a);
    printf("sum_cols ... \n");
#endif

#ifdef SUM_ROWS
    printf("sum_rows ... \n");
    int sum = sum_array_rows(a);
#endif

    printf("sum = %d\n", sum);

    return 0;
}
```

makefile

```
all: sum_rows sum_cols

sum_rows: sum.c
	gcc -DSUM_ROWS -o sum_rows sum.c

sum_cols: sum.c
	gcc -DSUM_COLS -o sum_cols sum.c

clean:
	rm sum_rows sum_cols
```

#COLS 10000 #ROWS 10000

```
user@DESKTOP-2UNLTD8:/mnt/c/Users/roseh/OneDrive/바탕 화면/대학 과제/2학년_2학기/시스템_프로그래밍/src/mem$ sudo perf st
at -e cache-misses ./sum_cols
[sudo] password for user:
sum_cols ...
sum = 1733793664

 Performance counter stats for './sum_cols':

          1609079      cache-misses

       0.543465429 seconds time elapsed

       0.387290000 seconds user
       0.148958000 seconds sys


user@DESKTOP-2UNLTD8:/mnt/c/Users/roseh/OneDrive/바탕 화면/대학 과제/2학년_2학기/시스템_프로그래밍/src/mem$ sudo perf st
at -e cache-misses ./sum_rows
sum_rows ...
sum = 1733793664

 Performance counter stats for './sum_rows':

        161717689      cache-misses

       1.312716627 seconds time elapsed

       1.116584000 seconds user
       0.189420000 seconds sys
```

프로그램을 실행하며 실행시간과 cache-misses 의 발생 빈도를 측정한다. 결과가 어떻게 나오는가? 그 이유를 써 보시오.

프로그램을 실행하며 실행시간과 cache-misses 의 발생 빈도를 측정한다. 결과가 어떻게 나오는가? 그 이유를 써 보시오.
- sum_array_cols () : 하나의 행에 있는 열의 값을 모두 더한 후 다음 행으로 이동
- sum_array_rows () : 하나의 열에 있는 행의 값을 모두 더한 후 다음 열으로 이동

sum_cols의 코드를 살펴보면 a[i][j]에서 1번째 루프에서 i에 접근 후 2번째 루프에서 a[i]에 대한 모든 원소에 접근이 끝나면 i++을 통해 다음 행(배열)로 넘어가는 형태를 띄며
sum에 대한 Temporal locality와 a에 대한 Spatial locality도 좋은 반면

sum_rows의 코드를 살펴보면 a[i][j]에서 1번째 루프에서 j를 정하고 2번쨰 루프에서 a[i]를 매번 옮기며 a[i]의 j 번째 원소에 접근하기 때문에 Spatial locality가 나쁘고 이로 인해 sum cols에 비해 cache-misses가 훨씬 높고 실행 시간도 높다.

#COLS 100000 #ROWS 10000

```
user@DESKTOP-2UNLTD8:/mnt/c/Users/roseh/OneDrive/바탕 화면/대학 과제/2학년_2학기/시스템_프로그래밍/src/mem$ sudo perf st
at -e cache-misses ./sum_cols
sum_cols ...
sum = 158067456

 Performance counter stats for './sum_cols':

          21654496      cache-misses

       6.463989575 seconds time elapsed

       4.557539000 seconds user
       1.898974000 seconds sys


user@DESKTOP-2UNLTD8:/mnt/c/Users/roseh/OneDrive/바탕 화면/대학 과제/2학년_2학기/시스템_프로그래밍/src/mem$ sudo perf st
at -e cache-misses ./sum_rows
sum_rows ...
sum = 158067456

 Performance counter stats for './sum_rows':

        3222343284      cache-misses

      12.992081585 seconds time elapsed

      11.365799000 seconds user
       1.588015000 seconds sys
```

프로그램에서 COLS 를 10배 증가시킨 후 perf 의 결과를 다시 추출한다. 결과가 어떻게 나오는가? 그 이유를 써 보시오.

위 문제에서 언급한 것과 같이 sum_cols의 코드를 살펴보면 sum에 대한 Temporal locality와 a에 대한 Spatial locality도 좋은 반면
sum_rows의 코드를 살펴보면 a에 대한 Spatial locality가 나쁘기 때문에 이로 인해 sum cols에 비해 cache-misses가 훨씬 높고 실행 시간도 높았다.

cols를 증가시키면 sum_cols와 sum_rows 둘 다 a의 모든 원소에 접근해야 하기 때문에 cache-misses가 증가했다. 다만 접근해야 하는 cols가 증가했기 때문에 매 루프마다 cols를 옮겨야 하는 나쁜 Spatial locality를 갖고 있는 sum_rows의 cache-misses 증가량이 1번째 루프에서 cols를 고정하고 원소에 접근하는 sum_cols의 cache-misses 증가량보다 훨씬 높은 것을 확인할 수 있다.

또한 sum의 결과가 오히려 cols가 10000일 때보다 줄어든 것을 확인할 수 있는데
int 자료형은 4바이트(32비트)기 때문에 표현할 수 있는 양수 최댓값은 2^31-1인 2,147,483,647인데
cols가 10000일 때 결과가 1,733,793,664로 최댓값에 거의 도달했기 때문에 더하기 연산을 더 수행하면 오버플로우가 난다. 원래 오버플로우가 1번만 일어나면 부호 비트가 1로 바뀌면서 음수 값이 나와야 하는데 더하기 연산을 너무 많이 수행해서 오버플로우가 2번 일어나며 다시 부호 비트가 0으로 바뀌어서 기존보다 작은 양수 값이 나왔다.