HIGH PERFORMANCE COMPUTING WITH COMMODITY HARDWARE
LAB 2

Kshitij Srivastava
UD ID - 702359111

Abstract: This lab is divided into 3 problems and 4 files (mem1.c, mem2.c, prof1.c, prof2.c). In the first problem, we have found out L1 and L2 cache misses as well as TLB misses. In the second problem, we have done node profiling and edge profiling and based on these two profiling we have designed CFG for prof1.c and prof2.c. Third problem is optimization problem where we were supposed to increase or decrease cache misses as given in the question

Machine Specification: Penryn machine was used.

PROBLEM 1: Use the PAPI library to measure the L1 cache miss, the L2 cache miss and the TLB miss of the function "func" of the two programs.

	L1 cache miss	L2 cache miss	TLB miss
mem1.c	131040	235	197597
mem2.c	5892432	5253578	1034578

The change in the code that was made to calculate L1 cache miss in mem1.c and mem2.c:

```
int events[1] = {PAPI_L1_DCM};
```

The change in the code that was made to calculate L1 cache miss in mem1.c and mem2.c:

```
int events[1] = {PAPI_L2_DCM};
```

The change in the code that was made to calculate L1 cache miss in mem1.c and mem2.c:

```
int events[1] = {PAPI_TLB_DM};
```

Screenshots of codes are attached below:

Calculating L1 cache miss for mem1.c

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

static unsigned len = 16*(1 << 20);

void func(int * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i]++;
    }
}

int events[1] = {PAPI_L1_DCM}; /*PAPI_L1_DCM, PAPI_L2_DCM, PAPI_TLB_DM*/
int eventnum = 1;

int main()
{
    long long values[1];
    int eventset;
    int * a;


    if(PAPI_VER_CURRENT != PAPI_library_init(PAPI_VER_CURRENT)){
        printf("Can't initiate PAPI library!\n");
        exit(-1);
    }

    eventset = PAPI_NULL;
    if(PAPI_create_eventset(&eventset) != PAPI_OK){
        printf("Can't create eventset!\n");
        exit(-3);
    }
    if(PAPI_OK != PAPI_add_events(eventset, events, eventnum)){
        printf("Can't add events!\n");
        exit(-4);
    }

    a = (int *) malloc(len*sizeof(int));
    PAPI_start(eventset);
    func(a);
    PAPI_stop(eventset, values);
    free(a);

    /*Print out PAPI reading*/
    char event_name[PAPI_MAX_STR_LEN];
    if (PAPI_event_code_to_name( events[0], event_name ) == PAPI_OK)
        printf("%s: %lld\n", event_name, values[0]);

    return EXIT_SUCCESS;
}
```

Calculating L2 cache miss for mem1.c

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

static unsigned len = 16*(1 << 20);

void func(int * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i]++;
    }
}

int events[1] = {PAPI_L2_DCM}; /*PAPI_L1_DCM, PAPI_L2_DCM, PAPI_TLB_DM*/
int eventnum = 1;

int main()
{
    long long values[1];
    int eventset;
    int * a;


    if(PAPI_VER_CURRENT != PAPI_library_init(PAPI_VER_CURRENT)){
        printf("Can't initiate PAPI library!\n");
        exit(-1);
    }

    eventset = PAPI_NULL;
    if(PAPI_create_eventset(&eventset) != PAPI_OK){
        printf("Can't create eventset!\n");
        exit(-3);
    }
    if(PAPI_OK != PAPI_add_events(eventset, events, eventnum)){
        printf("Can't add events!\n");
        exit(-4);
    }

    a = (int *) malloc(len*sizeof(int));
    PAPI_start(eventset);
    func(a);
    PAPI_stop(eventset, values);
    free(a);

    /*Print out PAPI reading*/
    char event_name[PAPI_MAX_STR_LEN];
    if (PAPI_event_code_to_name( events[0], event_name ) == PAPI_OK)
        printf("%s: %lld\n", event_name, values[0]);

    return EXIT_SUCCESS;
}
```

Calculating TLB miss for mem1.c

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

static unsigned len = 16*(1 << 20);

void func(int * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i]++;
    }
}

int events[1] = {PAPI_TLB_DM}; /*PAPI_L1_DCM, PAPI_L2_DCM, PAPI_TLB_DM*/
int eventnum = 1;

int main()
{
    long long values[1];
    int eventset;
    int * a;


    if(PAPI_VER_CURRENT != PAPI_library_init(PAPI_VER_CURRENT)){
        printf("Can't initiate PAPI library!\n");
        exit(-1);
    }

    eventset = PAPI_NULL;
    if(PAPI_create_eventset(&eventset) != PAPI_OK){
        printf("Can't create eventset!\n");
        exit(-3);
    }
    if(PAPI_OK != PAPI_add_events(eventset, events, eventnum)){
        printf("Can't add events!\n");
        exit(-4);
    }

    a = (int *) malloc(len*sizeof(int));
    PAPI_start(eventset);
    func(a);
    PAPI_stop(eventset, values);
    free(a);

    /*Print out PAPI reading*/
    char event_name[PAPI_MAX_STR_LEN];
    if (PAPI_event_code_to_name( events[0], event_name ) == PAPI_OK)
        printf("%s: %lld\n", event_name, values[0]);

    return EXIT_SUCCESS;
}
```

Calculating L1 cache miss for mem2.c

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

static unsigned len = 16*(1 << 20);

typedef struct Mem {
    int fa;
    char fb;
    int fc;
    char fd;
    int fe;
} Mem;

void func(Mem * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i].fa = a[i].fb+a[i].fd;
    }

    for(i=0; i<len; i++){
        a[i].fc = a[i].fe*2;
    }

}

int events[1] = {PAPI_L1_DCM}; /*PAPI_L1_DCM, PAPI_L2_DCM, PAPI_TLB_DM*/
int eventnum = 1;

int main()
{
    long long values[1];
    int eventset;
    Mem * a;

    if(PAPI_VER_CURRENT != PAPI_library_init(PAPI_VER_CURRENT)){
        printf("Can't initiate PAPI library!\n");
        exit(-1);
    }

    eventset = PAPI_NULL;
    if(PAPI_create_eventset(&eventset) != PAPI_OK){
        printf("Can't create eventset!\n");
        exit(-3);
    }
    if(PAPI_OK != PAPI_add_events(eventset, events, eventnum)){
        printf("Can't add events!\n");
        exit(-4);
    }

    a = (Mem *) malloc(len*sizeof(Mem));
    PAPI_start(eventset);
    func(a);
    PAPI_stop(eventset, values);
    free(a);
```

Calculating L2 cache miss for mem2.c

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

static unsigned len = 16*(1 << 20);
typedef struct Mem {
    int fa;
    char fb;
    int fc;
    char fd;
    int fe;
} Mem;

void func(Mem * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i].fa = a[i].fb+a[i].fd;
    }

    for(i=0; i<len; i++){
        a[i].fc = a[i].fe*2;
    }

}
int events[1] = {PAPI_L2_DCM}; /*PAPI_L1_DCM, PAPI_L2_DCM, PAPI_TLB_DM*/
int eventnum = 1;

int main()
{
    long long values[1];
    int eventset;
    Mem * a;

    if(PAPI_VER_CURRENT != PAPI_library_init(PAPI_VER_CURRENT)){
        printf("Can't initiate PAPI library!\n");
        exit(-1);
    }

    eventset = PAPI_NULL;
    if(PAPI_create_eventset(&eventset) != PAPI_OK){
        printf("Can't create eventset!\n");
        exit(-3);
    }
    if(PAPI_OK != PAPI_add_events(eventset, events, eventnum)){
        printf("Can't add events!\n");
        exit(-4);
    }

    a = (Mem *) malloc(len*sizeof(Mem));
    PAPI_start(eventset);
    func(a);
    PAPI_stop(eventset, values);
    free(a);
```

Calculating TLB miss for mem2.c

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

static unsigned len = 16*(1 << 20);

typedef struct Mem {
    int fa;
    char fb;
    int fc;
    char fd;
    int fe;
} Mem;

void func(Mem * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i].fa = a[i].fb+a[i].fd;
    }

    for(i=0; i<len; i++){
        a[i].fc = a[i].fe*2;
    }

}

int events[1] = {PAPI_TLB_DM}; /*PAPI_L1_DCM, PAPI_L2_DCM, PAPI_TLB_DM*/
int eventnum = 1;

int main()
{
    long long values[1];
    int eventset;
    Mem * a;

    if(PAPI_VER_CURRENT != PAPI_library_init(PAPI_VER_CURRENT)){
        printf("Can't initiate PAPI library!\n");
        exit(-1);
    }

    eventset = PAPI_NULL;
    if(PAPI_create_eventset(&eventset) != PAPI_OK){
        printf("Can't create eventset!\n");
        exit(-3);
    }
    if(PAPI_OK != PAPI_add_events(eventset, events, eventnum)){
        printf("Can't add events!\n");
        exit(-4);
    }

    a = (Mem *) malloc(len*sizeof(Mem));
    PAPI_start(eventset);
    func(a);
```

Problem 3:

3.1 Transform mem1.c so that the "func" functions of the programs can achieve maximum cache misses. Do the “de-optimization” for L2 cache miss and TLB miss. The "func" is basically a sequence of memory accesses. You can change the order of the memory accesses, but you cannot add or remove memory accesses to the sequence. Submit your code and measurements, together with your explanation of the transformations, i.e., why they work.

Mem1.c	Before De-optimization	After De - Optimization
L2 cache miss	235	15759928
TLB miss	197597	17284606

De - optimizing L2 cache for maximum misses:

Just as we discussed in Lab 1, if the stride of accessing numbers of array is in the multiple of 16, then it will always miss the cache because a cache line size is of 64 bytes and each integer occupies of 4 bytes of memory, therefore, only 16 integers can be accommodated in each line i.e. from 0 to 15.

It can be seen from observation that after using a stride length of 16, no of cache misses is 15759928 which is almost equal to 16M(length of array)

I even increased the stride length in multiples of 16 such as 32 , 64 , 128 ,256 and the cache misses were as follows:

Length/misses	32	64	128	256
Cache misses	16682859	16748587	16783473	16797612

A small portion of the code where the changes were made:

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

static unsigned len = 16*(1 << 20);

void func(int * a)
{
    int i,j;

    for(i=0; i<16; i++){
    for (j=i;j<=(len-16+i);j+=16){
    a[j]++;
    }
    }
}

int events[1] = {PAPI_L2_DCM}; /*PAPI_L1_DCM, PAPI_L2_DCM, PAPI_TLB_DM*/
int eventnum = 1;
```

De-optimizing TLB for maximum misses:

TLB works like any other cache and TLB has a page size of 4KB and has 256 entries. So, if we access array in multiples of 1024, then we will attain maximum TLB misses. The reason we do this is to access those numbers which are farther away from each other on memory and thus a copy of memory address is not available in TLB. However, we see that total TLB misses is greater than 16M and this is due to the fact because TLB can have both data and instruction misses.

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

static unsigned len = 16*(1 << 20);

void func(int * a)
{
    int i,j;

    for(i=0; i<1024; i++){
    for (j=i;j<=(len-1024+i);j+=1024){
    a[j]++;
    }
    }
}
```

3.2 Redesign the struct “Mem” in mem2.c to minimize the L2 cache miss. You may use struct-of- array, array-of-struct, or any combination. You CANNOT change the structure of the code in the function “func”, NOR change the order of operations. You can only revise the references to the variables depending on your design of the struct. Explain why your new design will minimize the L2 cache miss.

A) There were two ways of doing this problem. In the first way, I changed the order in which the variables were declared. Variables which were used in first ‘for’ loop were declared together, so that they can be present in a contiguous memory location. This has led to 20% increase in cache hit.

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

static unsigned len = 16*(1 << 20);

typedef struct Mem {
    int fa;
    char fb;
    char fd;
    int fc;
    int fe;
} Mem;

void func(Mem * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i].fa = a[i].fb+a[i].fd;
    }

    for(i=0; i<len; i++){
        a[i].fc = a[i].fe*2;
    }
```

Mem2.c	Before optimization	After optimization
L2 cache miss	5253578	4202879

B) The other way is to redesign the struct and this is how I have done it.

```
#include <papi.h>
#include <stdio.h>
#include <stdlib.h>
#define N (16*(1 << 20))
static unsigned len = 16*(1 << 20);

typedef struct Mem {
    int fa[N];
    char fb[N];
    char fd[N];
    int fc[N];
    int fe[N];
} Mem;

void func(Mem * a)
{
    int i;
    int j;
    for(j=0;j<N;j++){
        a[0].fa[j] = a[0].fb[j]+a[0].fd[j];
    }

    for(j=0;j<N;j++){
        a[0].fc[j] = a[0].fe[j]*2;

    }
```

Mem2.c	Before optimization	After optimization
L2 cache miss	5253578	831

The main reason as to why there is significance performance difference is because every corresponding element of the array fa, fb, fd, fc, fe is present in cache unlike in previous case. Even when the cache lines are loaded with next set of data, then also its kind of a synchronous process where corresponding elements between all the arrays are loaded at once in the cache.

3.3 Co-optimize the struct “Mem” and the function “func” to minimize the L2 cache miss. The only constraint is that your new code should implement the same workload, i.e., the same operations in the original code, and use the same amount of data. You can change the order of operations as well as the struct “Mem” in any way you want. Explain why your new design will minimize the L2 cache miss.

Code given in 3.2 (B) is structure of array. For this type of configuration, that’s the best combination of struct and func possible. It cannot be further optimized.

However, following code is Array of structure. In the following code, I changed the order of variables according to ‘for’ loop and because both the ‘for’ loops are independent of each other, I have merged them into one.

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

static unsigned len = 16*(1 << 20);

typedef struct Mem {
    int fa;
    char fb;
    char fd;
    int fc;
    int fe;
} Mem;

void func(Mem * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i].fa = a[i].fb+a[i].fd;
        a[i].fc = a[i].fe*2;
    }

}
```

Mem2.c	Before optimization	After optimization
L2 cache miss	5253578	380

The reason for getting such a low cache misses is because all the variables required in for loop are present in contiguous memory location.

I tried another version which is array of struct of array, which was not very good as compared to other two methods mentioned above but it is worth mentioning

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

static unsigned len = 16*(1 << 20);

typedef struct Mem {
    int fint[3];
    char fchar[2];
} Mem;

void func(Mem * a)
{
    int i;

    for(i=0; i<len; i++){
        a[i].fint[0] = a[i].fchar[0]+a[i].fchar[1];
    }

    for(i=0; i<len; i++){
        a[i].fint[1] = a[i].fint[2]*2;
    }

}
```

Mem2.c	Before optimization	After optimization
L2 cache miss	5253578	4202938

PROBLEM 2

In this problem, we were supposed to do node and edge profiling for prof1.c and prof2.c

For prof1.c :

NODE PROFILING

INPUT1
Counter at Node A: 10,000
Counter at Node B: 5040
Counter at Node C: 4960
Counter at Node D: 2556

Counter at Node E: 2484
Counter at Node F: 10000
Counter at Node G: 5006
Counter at Node H: 4994

INPUT2
Counter at Node A: 10,000
Counter at Node B: 8041
Counter at Node C: 1959
Counter at Node D: 4846
Counter at Node E: 3195
Counter at Node F: 10000
Counter at Node G: 2034
Counter at Node H: 7966

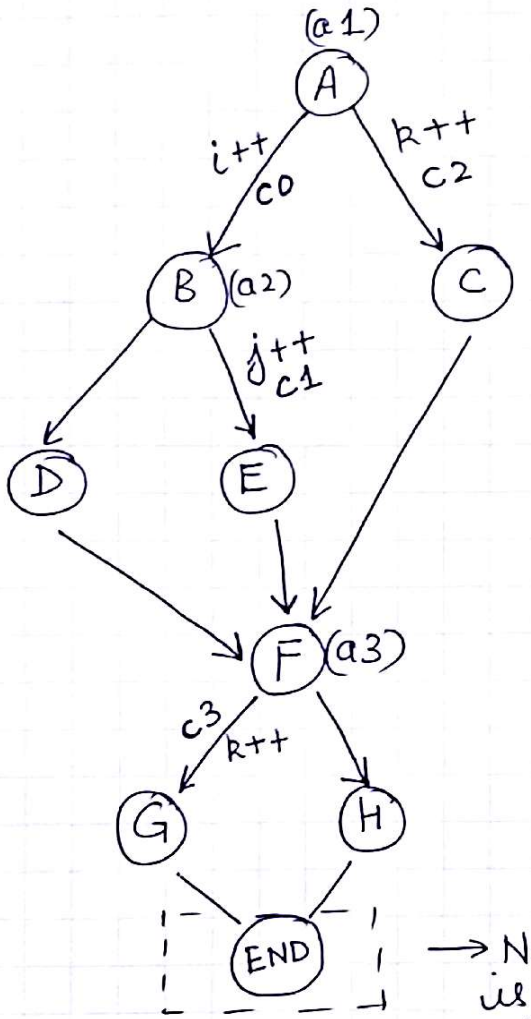
EDGE PROFILING

INPUT 1
Counter at AB: 4981
Counter at AC: 5019
Counter at BD: 2508
Counter at BE: 2473
Counter at CF: 5019
Counter at DF: 2508
Counter at EF: 2473
Counter at FG: 5085
Counter at FH: 4915

INPUT2
Counter at AB: 8032
Counter at AC: 1968
Counter at BD: 4862
Counter at BE: 3170
Counter at CF: 1968
Counter at DF: 4862
Counter at EF: 3170
Counter at FG: 2019
Counter at FH: 7981

PROBLEM 2 NODE PROFILING AND EDGE PROFILING.

PROF1.C



→ Not present in actual code but is used to show that A = end (G+F).

NODE

A → c0 + c2
B → c0
C → c2
D → c0 - c1
E → c1
F → c0 + c2
G → c3
H → c0 + c2 - c3

EDGE

A B → c0
A C → c2
B D → c0 - c1
B E → c1
C F → c2
D F → c0 - c1
E F → c1
F G → c3
F H → c0 + c2 - c3

For prof2.c :

NODE PROFILING

INPUT1
Counter at Node A: 10,000
Counter at Node B: 4965
Counter at Node C: 5035
Counter at Node D: 4931
Counter at Node E: 2488
Counter at Node F: 2581
Counter at Node G: 2455
Counter at Node H: 2476
Counter at Node I: 5069
Counter at Node J: 5069

INPUT2
Counter at Node A: 10,000
Counter at Node B: 8024
Counter at Node C: 1976
Counter at Node D: 6341
Counter at Node E: 3262
Counter at Node F: 397
Counter at Node G: 1885
Counter at Node H: 4456
Counter at Node I: 3659
Counter at Node J: 3659

EDGE PROFILING

INPUT 1
Counter at AB: 5059
Counter at AC: 4941
Counter at BD: 2492
Counter at BE: 2567
Counter at EI: 2567
Counter at CD: 2517
Counter at CF: 2424
Counter at FI: 2424
Counter at DG: 2495
Counter at DH: 2514
Counter at IJ: 4991

INPUT 2
Counter at AB: 8073

Counter at AC: 1927
Counter at BD: 4803
Counter at BE: 3270
Counter at EI: 3270
Counter at CD: 1539
Counter at CF: 388
Counter at FI: 388
Counter at DG: 1926
Counter at DH: 4416
Counter at IJ: 3658

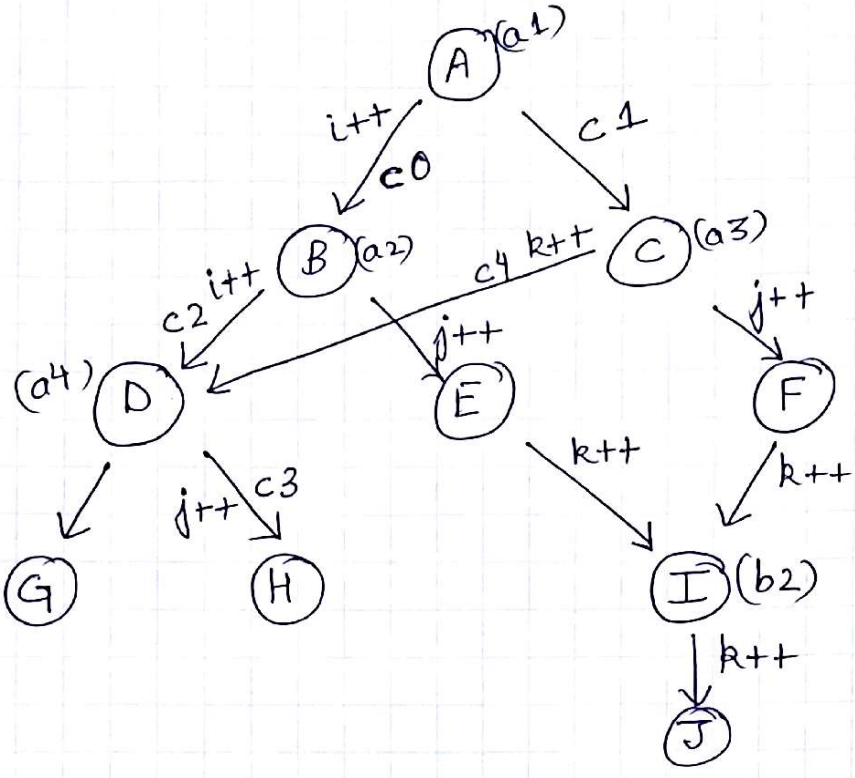
OPTIONAL ANSWER:

Path Profiling for Prof1.c

```
INPUT1
ABDFH:1223
ABDFG:1360
ABEFH:1216
ABEFG:1184
ACFG:2567
ACFH:2450

INPUT2
ABDFH:3832
ABDFG:1005
ABEFH:2475
ABEFG:614
ACFG:448
ACFH:1626
```

PROF 2. C



NODE

A → c0+c1
B → c0
C → c1
D → c2+c4
E → c0−c2
F → c1−c4
G → c2−c3+c4
H → c3
I → c0+c1−c2−c4
J → c0+c1−c2−c4

EDGE

A B → c0
A C → c1
B D → c2
B E → c0−c2
E I → c0−c2
C D → c4
C F → c1−c4
F I → c1−c4
D G → c2+c4−c3
D H → c3
I J → c0+c1−c2−c4